# VS Commands Summary

This is a condensed reference for VS Commands that may be used in any VS Solver. The full reference is the *VS Commands Reference Manual*. In following forms, [square brackets] indicate optional items.

## Setting Values

A parameter or variable is assigned a value with a statement with the form:

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

In this case, *expression* is evaluated immediately.

### *Expressions*

Table 1 lists the mathematical operators that may be used in an expression, in the order of precedence. Operators in the same row have the same precedence. Operators with the same precedence are evaluated left to right. Table 2 lists math functions that may be used in expressions.

*Table 1. Operator precedence.*

| Operator | Description |
|---|---|
| `()` | expressions in parentheses are evaluated first |
| `x^y` | Exponentiation: $x^y$ |
| `-x` | Unary minus |
| `~x` | Unary logical not (result is 1 if x is 0, else 0) |
| `x * y`<br>`x / y`<br>`x % y` | Multiplication, division, and modulo |
| `x + y`<br>`x - y` | Addition and subtraction |
| `x == y`<br>`x < y`<br>`x <= y`<br>`x ~= y`<br>`x > y`<br>`x >= y` | Comparisons between *x* and *y*; result is 1 if the comparison is true, 0 if it is false |
| `x & y` | Result is 1 if both x and y are not 0, else 0 |
| `x \| y` | Result is 1 if either or y is not 0, else 0 |

### *Units*

If *expression* includes any names of symbols, then all calculations are made using internal SI units. If there are no symbols—only numbers—then the result is converted to the display units for the variable *keyword*.

*Table 2. Math functions supported in symbolic expressions.*

| Function | Description |
|---|---|
| `ABS(x)` | Absolute value |
| `ACOS(x), ASIN(x)` | Arc-cosine & Arc-sine (w. range check) |
| `ATAN(x)` | Arc-tan with result $\pm \pi/2$ |
| `ATAN2(y, x)` | Arc-tan($y/x$) with result $\pm \pi$ |
| `COS(x),COSH(x)` | Cosine and Hyp. Cosine functions |
| `EXP(x)` | $e^x$ |
| `FLOOR(x),CEIL(x)` | Largest integer $\le x$, Smallest integer $\ge x$ |
| `FIX(x)` | Truncate to integer closer to zero |
| `IF(x, y, z)` | If *x* is not 0, return *y;* otherwise return *z.* Between *y* and *z*, only the one used is evaluated. |
| `INT(x)` | Nearest integer to *x* |
| `INVERSE(f,y,x,i)` | Inverse access to configurable function |
| `LOG(x), LOG10(x)` | Natural log (base e) and base 10 log. |
| `MAX(x, y),MIN(x, y)` | Max./Min. of two arguments |
| `PARTIAL[2]` | derivative of a table w.r.t x or y. |
| `SIN(x),SINH(x)` | Sine and Hyp. Sine functions |
| `SIGN(x, y)` | If $y > 0$ Then $\|x\|$ Else $-\|x\|$ |
| `SQRT(x)` | Square root (with range check) |
| `TAN(x),TANH(x)` | Tangent and Hyp. Tangent functions |

New units may be installed with the command:

`define_units` *units scale*

where *units* is printed name of units (e.g., `cm`) and *scale* is the scale factor to convert to SI to *units* (e.g., 100).

### *Description*

If *description* is provided, the description for the parameter keyword shown in Echo files is replaced with the new text.

## Embedded Python

*Table 3. Embedded Python functions and commands.*

| Function/Command | Description |
|---|---|
| `RUN_PYTHON_STRING` | Execute Python string |
| `PYTHON(cond, cmd, signal, in, out)` | Call Python as a function. (equations) |

## Define New Variables

Define a new variable with a command that has the form:

*command variable* [[=] *expression* [; [*units*] [; [*desc*]]]]

Table 4 lists the commands for defining new variables. All commands for new variables are listed near the end of Echo files written after the commands were processed.

*Table 4. VS Commands for new variables.*

| Command | Action |
|---|---|
| `DEFINE_IMPORT` | Define a new Import variable. |
| `DEFINE_OUTPUT` | Define a new output variable. |
| `DEFINE_PARAMETER` | Define a new parameter. |
| `DEFINE_VARIABLE` | Define a new state variable. |
| `DELETE_VARIABLE` *var* | Delete variable *var* (any type) that was defined earlier. |
| `RESET_EXPORTS` | Disable all Export variables. |
| `RESET_IMPORTS` | Disable all Import variables. |
| `RESET_LIVE_ANI` | Disable all live animator vars. |

## VS Events

VS Events are typically managed on Windows using the Events library screen in the GUI. Underlying the screen are VS Commands listed in Table 5. When working from Linux, it is necessary to use the commands directly.

*Table 5. VS Commands for handling Events.*

| Command | Action |
|---|---|
| `DEFINE_EVENT` | Define new Event. |
| `DELETE_EVENTS_ID` *id* | Delete all existing Events that have a specified group ID *id*. |
| `RESET_EVENTS` | Clear all pending Events. |
| `SET_EVENT_ID` *id* | Set an integer ID that will be associated with new Events. |
| `STOP_RUN_NOW` | Stop the simulation run. |
| `WRITE_LOG` *text* | Write *text* to the log file. |

An Event is defined with the command:

`DEFINE_EVENT` *condition* [; [*pathname*]]

where *condition* is an expression that is evaluated at the end of every time step. If it is zero, nothing happens. Otherwise, the Event is "triggered." If *pathname* was provided, the solver loads the Parsfile and continues. If *pathname* was not included, then the run terminates.

## User-Defined Functions

Table 6 lists commands used to define new functions, which can have up to four arguments. Functions are listed in the Echo file after new variables.

*Table 6. Commands for defining a new function.*

| Command | Purpose |
|---|---|
| BEGIN_FUNCTION | Start new function definition |
| DEFINE_LOCAL | Define local variables |
| RETURN | Specify the function return value |
| END_FUNCTION | End the function definition |

The definition of a function goes as follows:

BEGIN_FUNCTION *name*(*arg1*, *arg2*, …) ; *description*
  DEFINE_LOCAL *var1, var2, …*
  <*equations*>
  RETURN *formula*
END_FUNCTION

## Add Equations

Add new equations with commands that has the form:

*command variable* [=] *expression* [;]

Table 7 lists the commands to add equations. "EOM" means built-in equations of motion. All equations added with these commands are listed near the end of the Echo file.

*Table 7. Commands to add equations to the model.*

| Command | When Applied |
|---|---|
| EQ_PRE_INIT | Once, after reading input Parsfiles |
| EQ_INIT | Once, after the initialization |
| EQ_INIT2 | Once, after writing the Echo file |
| EQ_IN | Every time step, before any EOM |
| EQ_DYN | Every time step, after kin EOM |
| EQ_OUT | Every time step, after all EOM |
| EQ_DIFFERENTIAL | Every time step, after the EQ_OUT equations. |
| EQ_SAVE | Every time step, just after the EQ_DIFFERENTIAL equations. |
| EQ_FULL_STEP | At the end of every full time step |
| EQ_END | Once, before writing the End file. |

*expression* is not evaluated during the command unless it is 100% numerical. Otherwise, *expression* is evaluated every time the equation is applied.

Equations that are applied every time step follow the sequence shown in Figure 1.

## Random Numbers

Table 8 lists "special" VS functions that use additional resources to produce random numbers.

*Table 8. Random number functions.*

| Function | Description |
|---|---|
| RAND (*x*) | Pseudo-random number 0 - 1.0. |
| GENSEED (*x*) | Truly random seed for use with SRAND. |
| SRAND (*x*) | Seed RAND with positive number *x* . |

## Configurable Functions

The syntax for using a Configurable Function in an expression is:

*function* (*col_var*, *row_var*, *index*)

where *function* is the root name of the Configurable Function, *col_var* and *row_var* are two expressions used as inputs to the function, and *index* indicates which dataset to use.

The Echo file always shows a keyword for the function that includes the configuration, e.g., FD_TABLE might be shown if the root name is FD. Even if there is only one independent variable, a value must be specified for *col_var* (it will be ignored). Even if there is only one dataset, *index* must be specified (use 1).

Define a new set of Configurable Functions with the command:

DEFINE_TABLE *name n*

where *name* is the root name of the new function, and *n* is the number of datasets available for the function.

## Linearization *(not used in SuspensionSim)*

The math model in the VS Solver may be linearized with the form:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$$
$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t)$$

where **x** is an array of state variables, **u** is an array of control variables, and **y** is an array of outputs.

Table 9 shows VS Commands to generate **A**, **B**, **C**, and **D** matrices for use in MATLAB.

*Table 9. VS Commands used in linearization.*

| Command | Action |
|---|---|
| LINEARIZE *path* | Linearize and write matrices to M-file *path* |
| LINEAR_SV *sv* | Add variable *sv* to x array |
| LINEAR_CONTROL *con* | Add variable *con* to u array |
| LINEAR_OUTPUT *out* | add variable *out* to y array |

## Back Up in Time *(not used in SuspensionSim)*

Table 10 lists commands (and one function) to save the state of the VS Math Model in memory at various times, and to go back to a saved state if some condition occurs.

*Table 10. Save and restore the model state.*

| Command | Action |
|---|---|
| RESET_STATES | Clear all saved states |
| RESTORE_STATE *t* | Restore the model to state at time *t* and continue the run |
| SAVE_STATE | Save the current model state. |
| SAVED_STATE_TIME (*t*) | Function: time for the last state saved before *t*. |
| START_SAVE_TIMER *t_int* | Save the model state at timer interval *t_int*. |
| STOP_SAVE_TIMER | Stop saving with a timer |

## Calculations at Each Time Step

Figure 1 shows a schematic timeline for the main calculations each time step. In the figure, ΔT is TSTEP for the Euler and AB-2 integration methods; it is TSTEP/2 for the AM and RK methods.

Please see the *VS Commands Reference Manual* for more details and for alternative timelines.

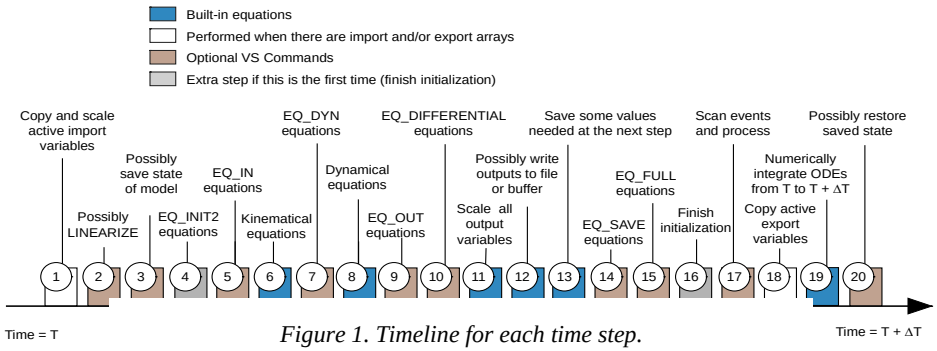


*Figure 1. Timeline for each time step.*