

# Connecting to External Tire Models with VS STI

Yukio Watanabe, Ph.D.

Overview of VS STI Co-Simulation .....	2
VS STI and Module .....	3
Tire Parameters .....	4
Source Code and Run Examples .....	4
Example: Connecting a TYDEX STI Tire Module .....	4
Exploring Example Datasets .....	5
Source Code for an Example VS STI Wrapper Module .....	7
Example: Connecting a Simple VS STI Tire Model .....	13
Exploring Example Datasets .....	13
Source Code for an Example VS STI Module .....	14
Reference: VS STI Functions .....	14
Functions in a VS STI Module .....	14
Callbacks from VS Solvers .....	20
References .....	23

In 1996, An interface called STI (Standard Tyre Interface) for connecting tire models to multibody vehicle models was published by a German working group called TYDEX [1]. Since then, methods implemented by the TYDEX STI have been used to connect tire and vehicle models, both commercial and in-house tools, in the world. However, the published standard is inadequate for connecting modern tire model tools to modern vehicle simulation models for these reasons:

1. The TYDEX STI standard is specific to the Fortran programming language and does not support modern programs written in C/C++.
2. The TYDEX STI standards support the use of arrays of parameters shared by the vehicle and tire model, but does not specify the contents of the arrays.

In VehicleSim products released prior to 2016, existing interfaces between VS Solvers and external tire models (such as TASS MF-Tyre/MF-Swift, COSIN FTire, and some customer models) have each been customized. They all use an interface for C/C++ that is similar to the Fortran TYDEX interface. They all pass the same kinematical information from the vehicle model to the tire model, and receive the same force and moment information from the tire into the vehicle model. However, the C source code for the library routines used by the VS Solver were customized for each tire model in order to deal with additional parameters and variables that are exchanged.

The information paths between VS vehicle and the external tire models are now unified with a set of functions called *VS STI* (VehicleSim Standard Tire Interface) in version 2016 for CarSim/TruckSim, and in version 2017.1 for BikeSim. Program modules - Windows DLL files or dSPACE DS1006 library files - that use VS STI are called *VS STI Modules*.

This document is a reference for the VS STI functions. It also describes examples provided with BikeSim, CarSim, and TruckSim that include C source code and Microsoft Visual Studio project files.

<b>Note</b>	The VS STI module can be compiled for a Windows DLL file or for a dSPACE DS1006 library file. All other platforms or operating systems are not yet supported as of May 2017.
-------------	--

## Overview of VS STI Co-Simulation

Figure 1 shows a diagram of how a VS vehicle model works together with a third-party tire model. The VS Solver simulates the entire vehicle except the wheel/tire (above the wheel spin axis) and the tire model simulates wheel/tire dynamics including tire/ground contact (below the wheel spin axis).

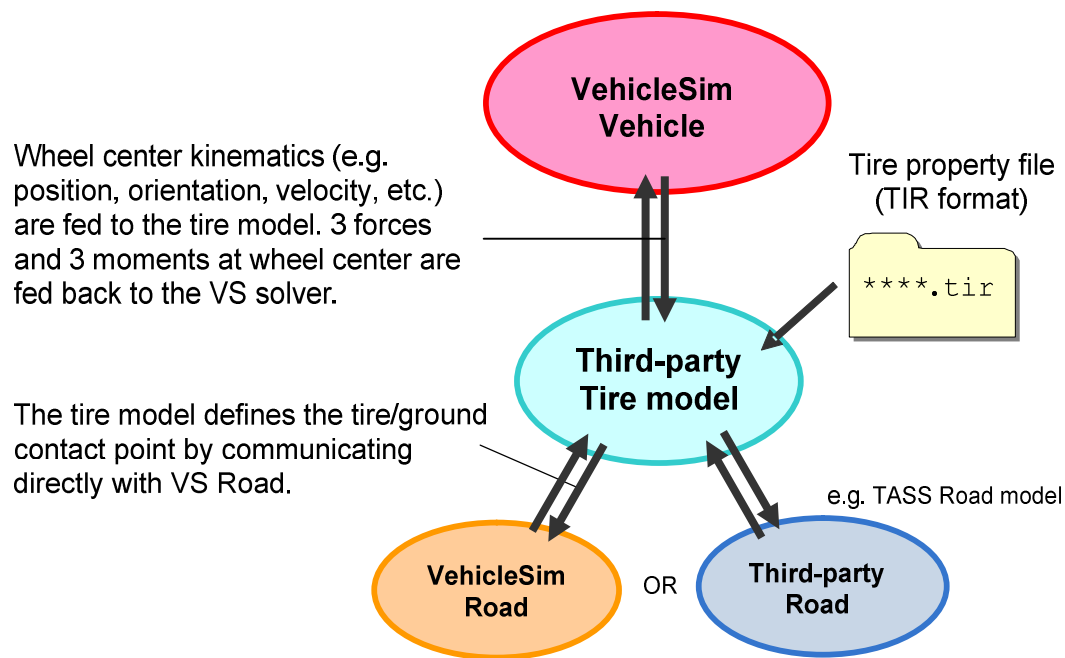


Figure 1. Co-simulation with a VS Solver and third-party tire model.

The VS Solver calculates kinematical variables at each wheel center such as position coordinates, velocity, orientation angle, angular speed, wheel rotation speed, and provides them to the tire model. The tire model calculates three forces and three moments at the wheel center that are sent back to the vehicle model.

The VS Solver provides road information as part of the VS API. As shown in Figure 1, third-party tire models can use the VS API road function to find the tire/ground contact point or contact surface. As an alternative to the VS road, you can also use functions within the third-party tire model.

## VS STI and Module

The VS vehicle model and external tire models are implemented with separate program files that communicate each other through the VS STI during the simulation. Figure 2 shows the program structure for interfacing between a VS Solver and external tire models, and corresponding program settings made from the **Tire (External)** screen.

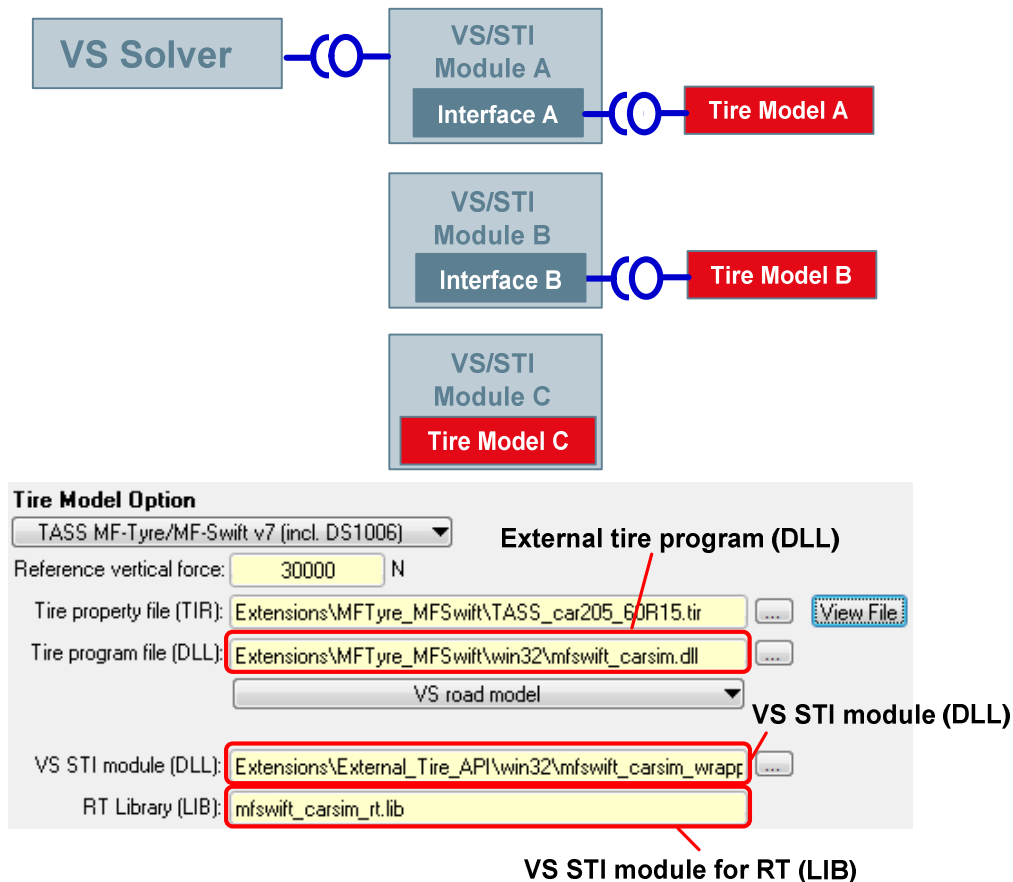


Figure 2. Connections between a VS Solver and tire model (upper) and corresponding settings on the Tire (External) screen (below).

The VS STI Module is a program file (e.g., Window DLL) that uses VS STI functions to work with a VS Solver. The VS STI Module can either act as an interface between the VS Solver and another external module with a custom tire model, or it can include the external tire model. Some sample programs and Microsoft Visual Studio projects are provided with BikeSim, CarSim, and TruckSim to show these options, and provide working C source code that can be customized as needed.

## Tire Parameters

Parameters for a tire model are typically specified in a separate file (.tir)> In this case, you will specify the parameter file pathname on the **Tire (External)** screen (Figure 2). Alternatively, the tire parameters can be specified VS table format (root keyword: STI\_TYPARR), in case the external model does not have a standard type of data file, or if the model will be run on an operating system that does not support standard file operations (e.g., the dSPACE DS1006). The external tire model either reads a TIR file directly or the VS Solver reads the STI\_TYPARR table and sends the array of parameters to the tire model depending on the cases.

## Source Code and Run Examples

All of the source code needed to build a VS STI module and several sample programs are contained in Extensions\User\_Tire\. The contents of folders under User\_Tire are summarized in Table 1.

*Table 1. Contents of User\_Tire folder*

Folder Name	Contents
Common	Header files that declare VS data types and API functions. These are used for the three projects.
Simple_sti	C source and project files for a simple tire TYDEX STI tire module.
vsStiWrapper	C source and project files for a sample VS STI “wrapper” module that connects a VS Solver to the example TYDEX STI tire module.
vsStiSimpleTire	C source and project files for a sample VS STI module that contains the same simple tire model as Simple_sti.

Files in each folder (except Common) are organized to create a DLL file with Microsoft Visual Studio Express 2013 for Windows. Each folder involves the following files with a project name.vcxproj and .filters (project files) and .sln (a “solution file” that identifies the project files).

Example runs are provided to make use of each of the two VS STI modules in BikeSim, CarSim and TruckSim.

If you intend to connect the VS Solvers with your a tire program that already exists (the tire model can be based on TYDEX STI or any other format), you can reference the example described in the next section.

If you intend to create your tire model from scratch and connect it with VS Solvers, you can reference the example described in the section **Example: Connecting a Simple VS STI Tire Model** (page 13).

## Example: Connecting a TYDEX STI Tire Module

This example shows how to connect VS Solvers to external tire models that exist as Windows DLL files. In this case, the VS STI module acts as an interface between the VS Solver and the external tire DLL (Figure 3).

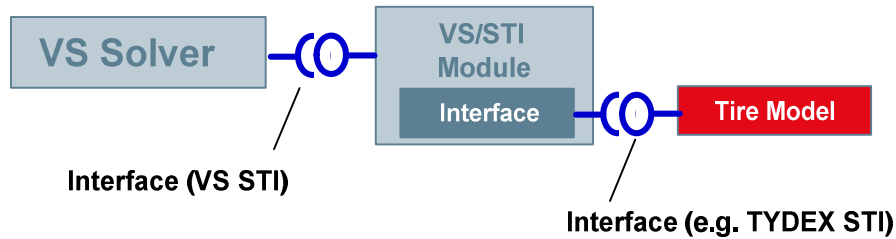


Figure 3. Interfacing between a VS Solver and external tire program.

## Exploring Example Datasets

This example uses a simple tire model which has a TYDEX STI. The example run in CarSim is named **Simple TYDEX/STI Model: DLC** in the category **\* CS 2016 - User Tire Model/STI** of the **Datasets** menu when viewed from the **Run Control** screen (Figure 4). TruckSim includes a similar example.

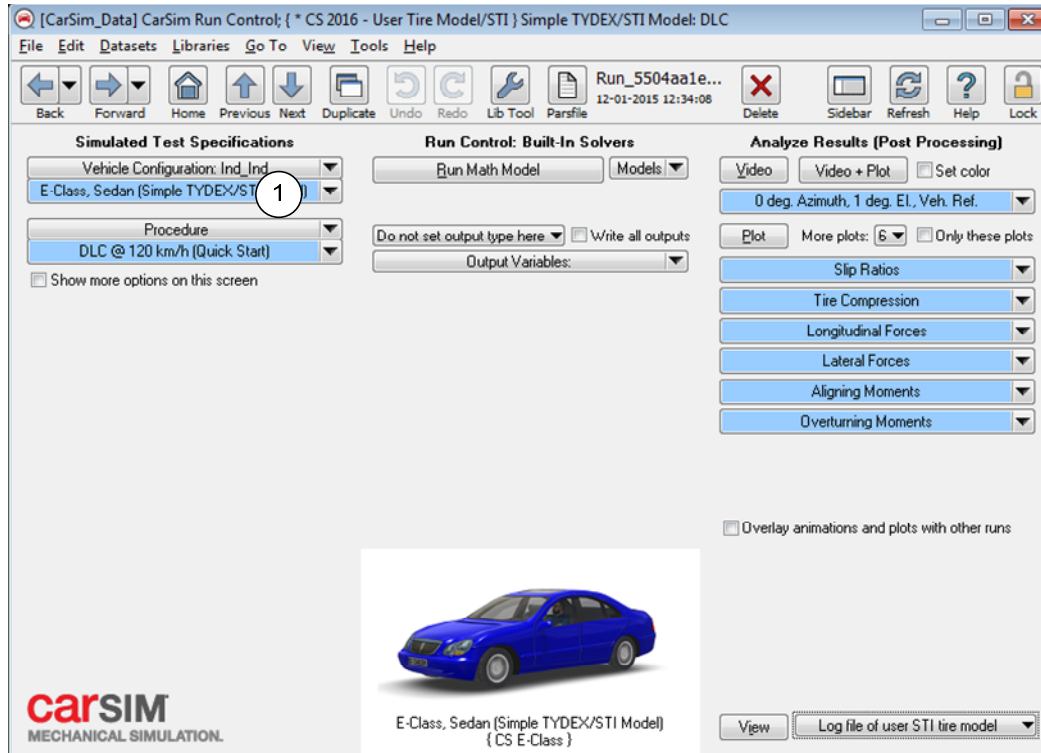


Figure 4. Run Control screen for Simple TYDEX/STI tire model.

Click the **Vehicle** link ① and then the **Tire** link to view the **Tire (External)** screen (Figure 5).

The specified tire program file (DLL) ① and VS STI module (DLL) ② are in the database folder `Extensions\User_Tire\`. Instead of reading the tire property file (TIR) ③, this tire model reads a tire parameter array from a tabular dataset (root keyword: `STI_TYPARR`) specified by a linked generic table dataset ④.

Click the **Parsfile** button (5) to view the keywords and information specified on the screen in a text file (Figure 6). Notice that the file path information for the tire program file (DLL) and VS STI module (DLL) are indicated by keywords **USER\_STI\_DLL** (1) and **EXTERNAL\_TIRE\_MODULE\_DLL** (2), respectively.

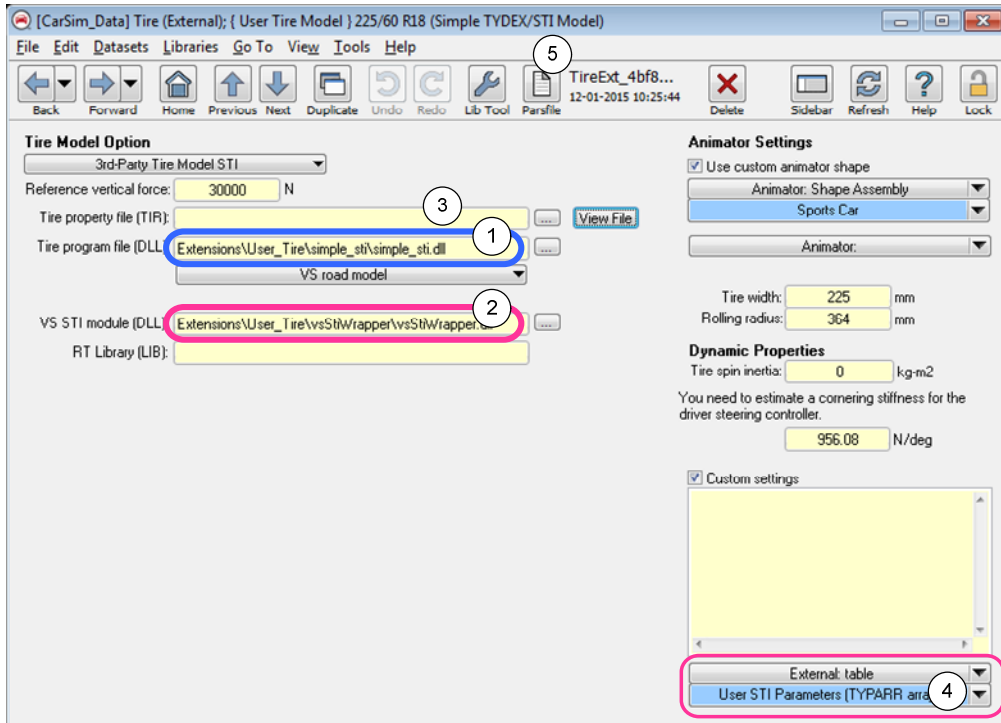


Figure 5. Setting of a simple TYDEX STI tire model on Tire (External) screen.

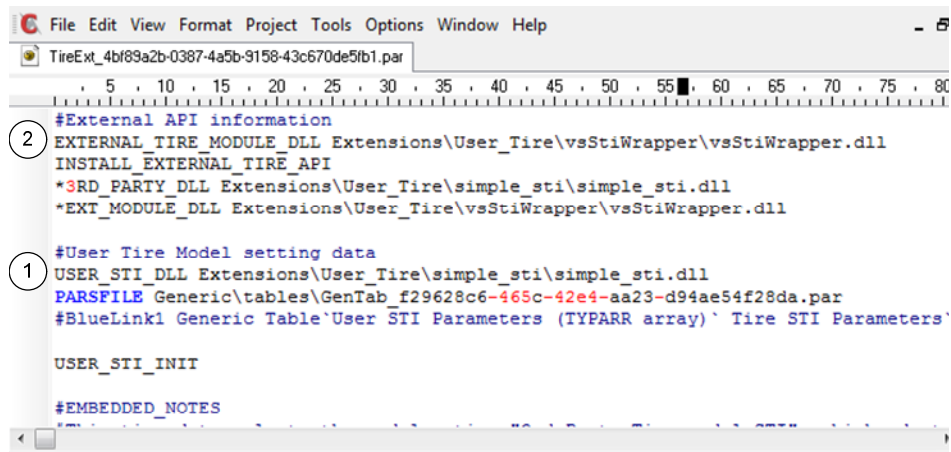


Figure 6. Parsfile written by Tire (External) screen

Click the **External: table** link (4) (Figure 5) to view the **Generic Table** screen (Figure 7).

On this screen, the function root keyword (1) must be set as **STI\_TYPARR**. The data graphic display (2) doesn't mean anything here. This simple tire model characterizes the forces and

moments in linear by using only 6 linear parameters specified in the data field (3). See the Notes field (4) for the definitions of these parameters.

By clicking the **Back** button (5) three times or clicking **Home** button (6), you return to the **Run Control** screen. Run this example, and observe the plot and animation (Figure 8).

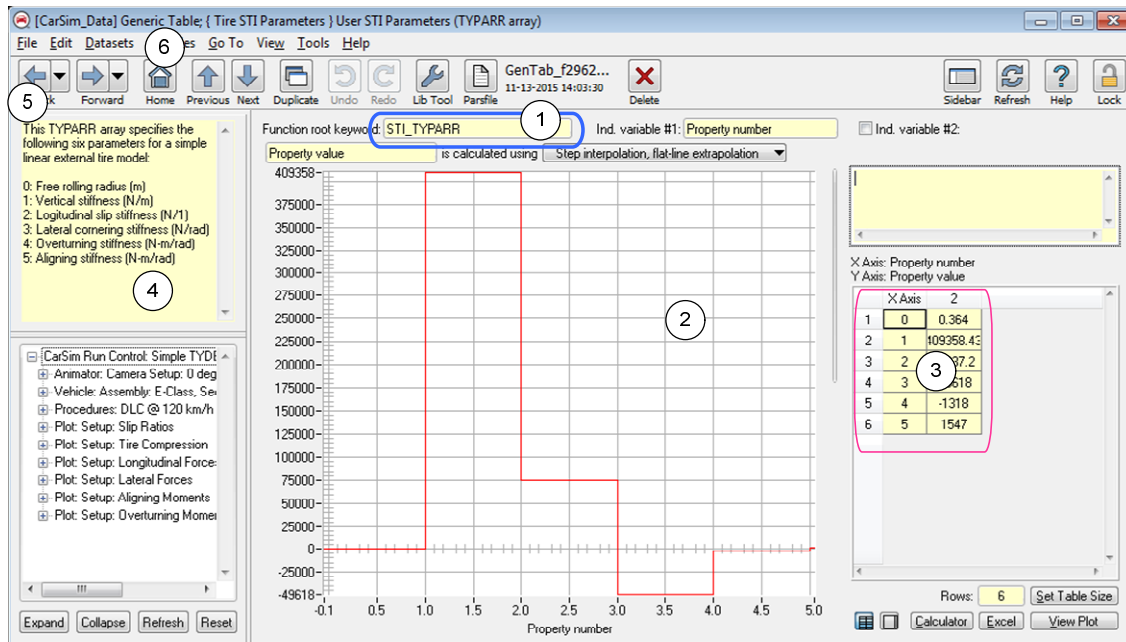


Figure 7. Setting of a tire parameter array on Generic Table screen

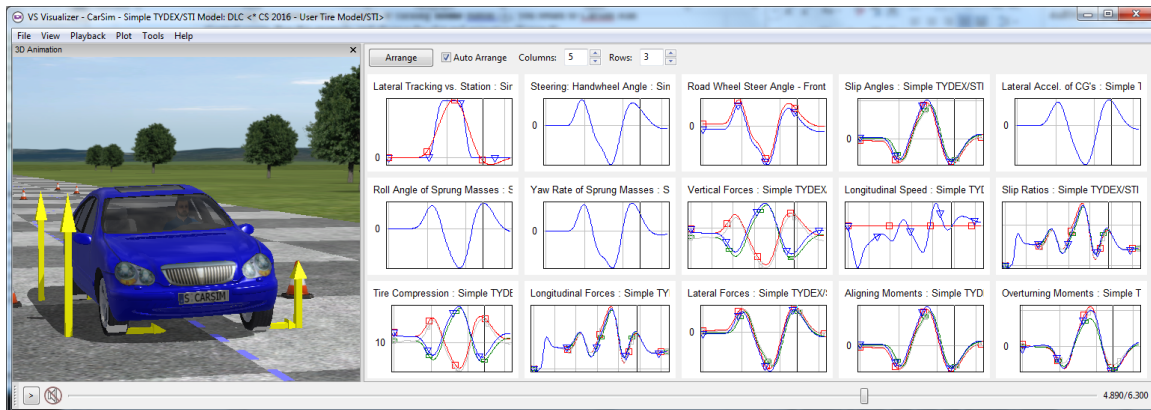


Figure 8. Simulation result with a simple linear tire model in animation and plot.

## Source Code for an Example VS STI Wrapper Module

The source code and project files for the VS STI module that is used for this example (specified on **Tire (External)** screen, VS STI module (2) shown in Figure 5) are contained in the folder Extensions\User\_Tire\vsStiWrapper\. The DLL created with this project will in turn load the simple example TYDEX STI DLL, which contains the tire model equations.

Figure 9 shows the source code and project as displayed for the wrapper within Visual Studio Express 2013.

```

21 typedef struct {
22     vs_real typarr[27], wrkarr[1], infVar[100], ropar[10],
23     dis[3], tramat[9], vel[3], omega[3], force[3], torque[3],
24     angtwc, omegar, deqvar, deqini, deqder;
25     int is_first, ntypar, ndeqvr, nropar, nwork, niwork, nvars,
26     iswtch, iwrkar[1], idtyre, ierr, nchtds, nchrds, idroad,
27     len_tyrmod, len_chrdst, len_chtdst, len_road;
28     char tyrmod[256], chrdst[256], chtdst[256], chsdst[256], road[256],
29     descVar[100][33], keyVar[100][15];
30     int id[30], // indices to all parameters, used to set visibility
31     npar, // number of parameters (129 the last time this was updated)
32     *opt_tire_model; // pointer to external tire model option parameter
33 } UserPars;
34
35 static UserPars **sUserTire;
36 static int sUserTireN;
37 static vs_real sTime;
38 static int *sIside, *sAxleMap, *sUnitsN, *sIunit, *sIaxle, *sItire;
39
40 static HMODULE USER_Module = NULL;
41 static int sDualsArePossible = FALSE; // generate labels for duals?
42 static int sIsDLLfound = FALSE;
43 static FILE *sUserLogFile;
44
45 #define READ_BUFFER_LEN 200
46
47 VS_STI_WRITE_TO_LOGFILE_FUNC gWrite_to_logfile;
48 VS_API_EXPORT void vs_sti_write_to_logfile(VS_STI_WRITE_TO_LOGFILE_FUNC function_r
49     gWrite_to_logfile = function_name;
50 }

```

Figure 9. Part of a C program for a VS STI Wrapper Module.

The variables are defined in the beginning of the program (lines 21 – 45).

Lines 47 – 128 install the callback functions that make some of VS functions (that are defined in the VS Solvers) become available on this program. Lines 130 through 313 (Figure 10) are miscellaneous local functions specifically used to echo the tire information.

VS STI functions used by VS Solvers to access the external tire model are listed in lines 315 through the end of the file. All functions named with prefix `vs_sti_` (for example `vs_sti_calc` and `vs_sti_install`) are accessed from VS Solvers. The details of the VS STI functions are described in a later reference section **Reference: VS STI Functions** (page 14).

Table 2 the main local functions in this program (except the miscellaneous functions for echo, lines 130 – 313).

The following subsections describe the sequence of VS STI function calls made by a VS Solver.



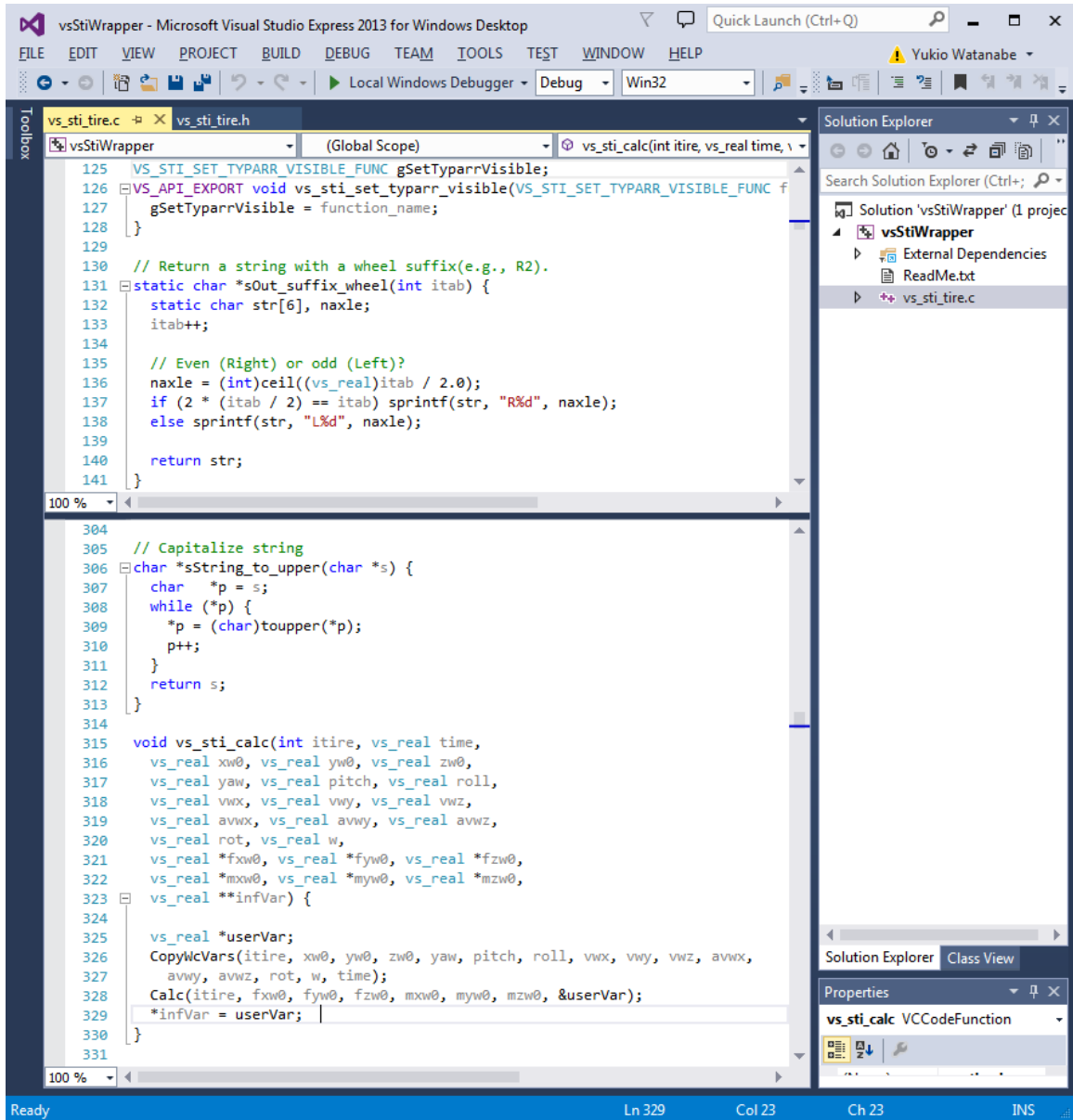


Figure 10. Miscellaneous local functions for the example VS STI Module.

### Installing the External Tire Program

When a VS Solver scans the input Parsfiles, it looks for the keyword `EXTERNAL_TIRE_MODULE_DLL`, which is followed by a pathname for an VS STI module. It also looks for the keyword `INSTALL_EXTERNAL_TIRE_API`. Those keywords are written in the parsfile of **Tire (External)** screen (Figure 6). When the keyword `INSTALL_EXTERNAL_TIRE_API` is found, the VS Solver loads the VS STI module and calls the function `vs_sti_install` from the tire DLL.

In this example, the function `vs_sti_install` call the local functions `sInstall` and `sSetdef`, which allocate memory for the tire model and set default values for parameters.

Table 2. Local functions supporting VS STI interface.

Function	Description
sCalc	Calls the TYDEX STI tire model function through sCallSti with the normal operation mode (job = 0).
sCallSti	Calls TYDEX STI tire model function (i.e. simple_sti)
sCopyWcVars	Copy the variables at wheel center with a single tire (fed from VS Solver) to the vectors feeding to TYDEX STI model.
sCopyWcVars2	Copy the variables at axle center with dual tires (fed from VS Solver) to the vectors feeding to TYDEX STI model.
sFree	Frees memory allocated by sInstall at the end of simulation.
sInit	Calls the TYDEX STI tire model function through sCallSti with an initial information (job = 2).
sInstall	Allocate memory and connect with parameters from the VS Solver.
sLoad	Load TYDEX STI tire model program (DLL) and get the address of simple_sti.
sSetdef	Define default values for parameters

### Loading and Initializing the External Tire Module

After the VS STI module is loaded by a VS Solver, the VS Solver calls `vs_sti_scan` to scan for other keywords related to the external tire model. In this example program, `vs_sti_scan` is programmed in lines 556 – 639 (Figure 11).

When a VS Solver detects the keyword `USER_STI_DLL`, it calls `sLoad` (line 577) to load the TYDEX STI DLL. If the keyword `USER_STI_INIT` is found, the function `sInit` (line 635) is called to initialize the tire model. The data keyword `USER_STI_INIT` specified for the external tire model should be the last one.

### Calling the External Tire Program

As the simulation runs, communication with the external tire model is always through `sCallSti` (Listing 1) during the initialization, normal operation, and termination of the simulation. The job number indicates the context for the calculations.

In this listing, `simple_sti` is the callback function from the TYDEX STI tire program. The argument *job* indicates the purpose of the call, with conventions such as:

- 0/5    normal calculation
- 1      initialization
- 2      reading parameters
- 99     termination

`sLoad` calls `sCallSti` with initialization mode (job = 1) while `sInit` calls `sCallSti` with reading parameter mode (job = 2).

For the normal tire calculation, `vs_sti_calc` calls `sCallSti` through the local function `sCalc` with normal calculation mode (job = 0) and at the end of simulation `vs_sti_exit` calls `sCallSti` with termination mode (job = 99).

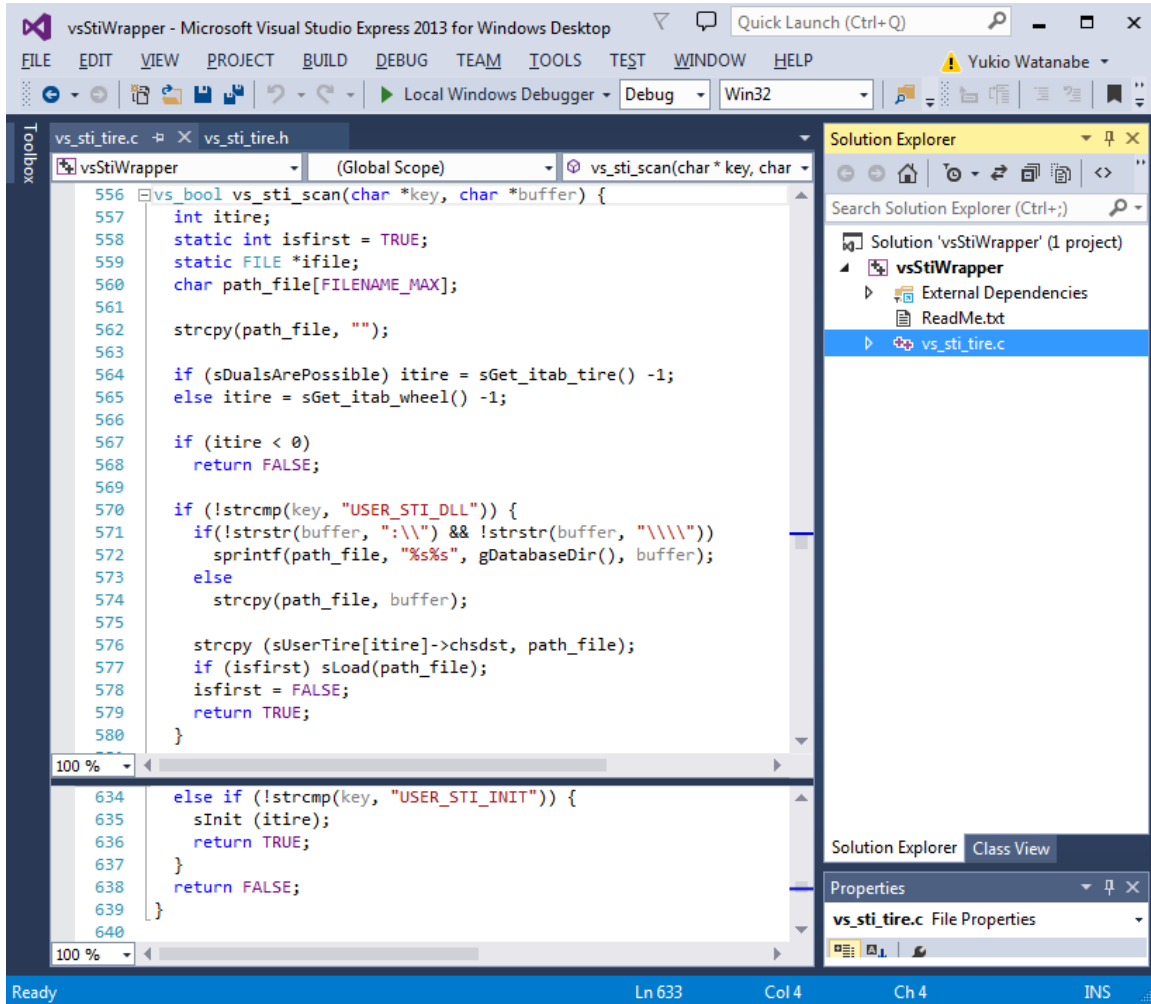


Figure 11. Keyword scanning routing, vs\_sti\_scan (for wrapper a TYDEX STI tire model).

### Calculation of Tire Forces and Moments

The VS Solvers call function vs\_sti\_calc (

Listing 2) to calculate the tire forces and moments. The VS Solver calculates kinematical variables at each wheel center such as position coordinates (xw0, yw0 and zw0), velocities (vwx, vwy and vwz), orientation angles (yaw, pitch and roll), angular speeds (avwx, avwy and avwz), wheel rotation angle and speed (rot and w), and provides them to the tire model. The tire model calculates three forces (fxw0, fyw0 and fzw0) and three moments (mxw0, myw0 and mzw0) at the wheel center that are sent back to the vehicle model.

A VS Solver calls vs\_sti\_calc every calculation time step with updated kinematical variables. The definitions of all kinematical variables are as same as TYDEX STI [1]. sCopyWcVars copies the variables in three directions (e.g. xw0, yw0 and zw0) to vector elements (e.g. dis) and orientation angles (yaw, pitch and roll) are converted to a transformation matrix (tramat) feeding to TYDEX STI function (i.e. simple\_sti).

*Listing 1. Source code for sCallSti in example.*

```
static void sCallSti (UserPars *mp, int *job, const char *suffix) {
    if (!sIsDLLfound) {
        gPrintf_error(
            "the solver program (DLL) was not found, not properly loaded, "
            "or not supported on this platform.");
        return;
    }

    simple_sti(sMessage, &mp->iswtch, job, &mp->idtyre, &sTime, mp->dis,
        mp->tramat, &mp->angtwc, mp->vel, mp->omega, &mp->omegar,
        &mp->ndeqr, &mp->deqvar, &mp->ntypar, mp->typarr, &mp->nchtds,
        mp->chtdst, &mp->len_chtdst, vscUserRoad, &mp->idroad,
        &mp->nropar, mp->ropar, &mp->nchrds, mp->chrds, &mp->len_chrds,
        mp->force, mp->torque, &mp->deqini, &mp->deqder,
        mp->tyrmod, &mp->len_tyrmod, &mp->nvars, mp->infVar, &mp->nwork,
        mp->wrkarr, &mp->niwork, mp->iwrkar, &mp->ierr);

    if (mp->ierr > 2)
        gPrintf_error("USERSTI had error (Job flag = %d, Tire %s, "
            "error code = %d) \n%s \n", *job, suffix, mp->ierr, mp->tyrmod);
}
```

*Listing 2. Source code for vs\_sti\_calc in example.*

```
void vs_sti_calc(int itire, vs_real time,
    vs_real xw0, vs_real yw0, vs_real zw0,
    vs_real yaw, vs_real pitch, vs_real roll,
    vs_real vwx, vs_real vwy, vs_real vwz,
    vs_real avwx, vs_real avwy, vs_real avwz,
    vs_real rot, vs_real w,
    vs_real *fxw0, vs_real *fyw0, vs_real *fzw0,
    vs_real *mxw0, vs_real *myw0, vs_real *mzw0,
    vs_real **infVar) {

    vs_real *userVar;
    sCopyWcVars(itire, xw0, yw0, zw0, yaw, pitch, roll, vwx, vwy, vwz, avwx,
        avwy, avwz, rot, w, time);
    sCalc(itire, fxw0, fyw0, fzw0, mxw0, myw0, mzw0, &userVar);
    *infVar = userVar;
}
```

### ***Echoing Parameters***

VS Solvers call `vs_sti_echo` at the beginning and end of simulation run. The information you specify in `vs_sti_echo` is written in echo files, `_ECHO.par` and `_END.par`.

### ***Terminating the External Tire Program***

VS Solvers call `vs_sti_exit` (shown below) at the end of simulation run. In this program, `vs_sti_exit` calls `sCallSti` with termination mode (`job = 99`) for proper ending of the tire calculation.

## Example: Connecting a Simple VS STI Tire Model

This example is intended for by users who intend to connect with a custom tire model that is not necessarily using functions specified in the old TYDEX STI. In this case, the VS STI module acts as an external tire program shown in Figure 12.

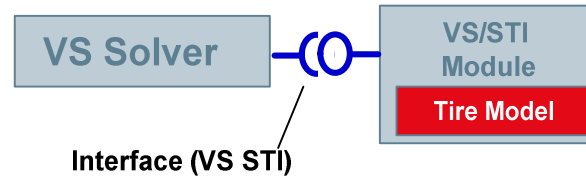
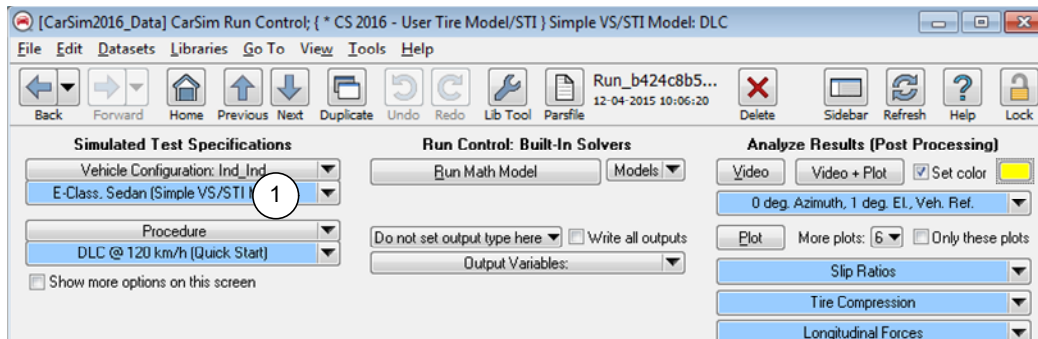


Figure 12. External tire program directly connected with VS Solver through VS STI.

## Exploring Example Datasets

This example uses a simple tire model which is implemented in a VS STI module. The simulation run in CarSim is named **Simple VS/STI Model: DLC** in the category **\* CS 2016 - User Tire Model/STI** of the **Run Control** screen (Figure 13). TruckSim includes a similar example.



Clicking **Vehicle** link ① and then clicking **Tire** link leads you the **Tire (External)** screen as shown in Figure 5.

The external tire program (DLL) ① does not exist in this example because the external tire model is implemented in the specified VS STI module (DLL) ② (in the folder `Extensions \User_Tire\vsStiSimpleTire\`). Instead of reading the tire property file (TIR) ③, this simple tire model reads a tire parameter array from a VS tabular dataset (root keyword: `STI_TYPARR`) specified by a generic table screen linked from ④. The tire model parameters are the exactly same as the other example and see the details in the previous chapter.

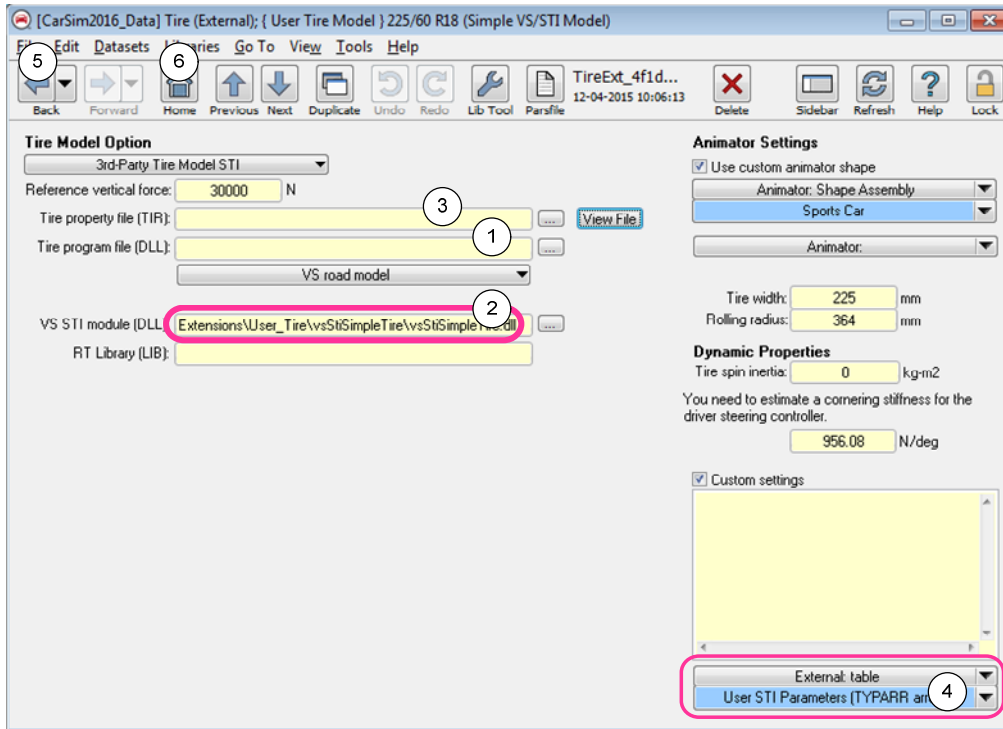


Figure 14. Setting of a simple VS STI tire model on Tire (External) screen.

By clicking the **Back** button (5) twice or clicking **Home** button (6), you return to **Run Control** screen. Run this example, and observe the plot and animation. The simulation results should be as same as seen (see Figure 8, page 7).

## Source Code for an Example VS STI Module

The source code and project files for the VS STI module used for this example are contained in the folder `Extensions\User_Tire\vsStiSimpleTire\`.

The program (`vs_sti_simple_tire.c`) is very similar to the other program (`vs_sti_tire.c`) used in the previous example. In this case, the source code for the simple TYDEX STI module is combined with the source code for the VS STI wrapper, to make a self-contained VS STI Module.

## Reference: VS STI Functions

This section documents the functions that make up the VS STI.

### Functions in a VS STI Module

Table 3 lists the VS STI functions which the VS Solvers call in the VS STI module to communicate with the external tire models.

Table 3. VS STI functions used to access the external tire models.

Function	When	Description
vs_sti_install	Read	Allocate local memory and set default values. VS Solvers call at the data scanning process to install the external tire program.
vs_sti_loaded_radius	Init	This function should returns a loaded tire radius in order to initialize axle heights of vehicle. VS Solvers call at the initialization process
vs_sti_scan	Read	Set the external tire parameters to be scanned by the VS Solvers. VS Solvers call at the data scanning process.
vs_sti_echo	Echo	VS Solvers call at the data echoing process to write the external tire data parameters to echo files.
vs_sti_SetVisible	Echo	Set the visibilities of the external tire data parameters.
vs_sti_calc	Run	Calculate the tire forces and moments by using kinematical values at the wheel center.
vs_sti_calc2	Run	Calculate the tire forces and moments for dual tires by using kinematical values at the axle center (each side).
vs_sti_exit	Exit	Set a termination process of the external tire model for proper exit the external tire program. VS Solvers call at the ending process.

**Note** All VS STI functions listed in Table 3 are called by VS Solvers. Therefore, all of them must exist, even if they contain no code.

### *vs\_sti\_install*

Declaration:

```
void vs_sti_install(int nTires, int duals);
```

This is the function called when a VS Solver detects the keyword `INSTALL_EXTERNAL_TIRE_API`. *nTires* is the number of tires in the vehicle model; *duals* indicates whether the model supports dual tires (is set to 0 in BikeSim/CarSim and 1 in TruckSim).

### *vs\_sti\_loaded\_radius*

Declaration:

```
vs_real vs_sti_loaded_radius(int itab, vs_real fz,
                             vs_real omegar);
```

Return:

Loaded radius of tire

The VS Solver calls this function during initialization. *itab* is a 1-indexed tire ID number, *fz* is the initial vertical load, and *omegar* is the initial wheel angular speed (rad/s).

### *vs\_sti\_scan*

Declaration:

```
vs_bool vs_sti_scan(char *key, char *buffer);
```

Return:

1 if the string *key* matches one of the external tire model parameters, or 0 if not.

When the VS Solvers read each line of the data file, they call this function the first word *key* and the rest of the line of text *buffer*.

If this function returns 0, the VS Solver keeps checking keyword by other scanning functions. If this function returns 1, the VS solver continues to read the next line of text from the input Parsfile.

For example, consider the code shown in Listing 3. Source code for example scanning.. Whenever the VS Solver reads the keyword “USER\_STI\_DLL”, it copies *buffer* to the string *path\_file*, possibly passes the value to the function *sLoad*, and then returns TRUE (= 1).

*Listing 3. Source code for example scanning.*

```
if (!strcmp(key, "USER_STI_DLL")) {
    if(!strstr(buffer, ":\\")) && !strstr(buffer, "\\\\"))
        sprintf(path_file, "%s%s", gDatabaseDir(), buffer);
    else
        strcpy(path_file, buffer);

    strcpy (sUserTire[itire]->chsdst, path_file);
    if (isfirst) sLoad(path_file);
    isfirst = FALSE;
    return TRUE;
}
return FALSE;
```

### *vs\_sti\_echo*

Declaration:

```
void vs_sti_echo(FILE *ifile, int itire);
```

VS Solvers call this function at the beginning and end of simulation run to write the external tire data parameters to echo files, *\_ECHO.par* and *\_END.par*, respectively. *ifile* is the pointer to the echo file (either *\_ECHO.par* or *\_END.par*) and *itire* is the tire number.

For example, when this function use the code shown in Listing 4, the information is displayed in the echo files as shown in Figure 15.

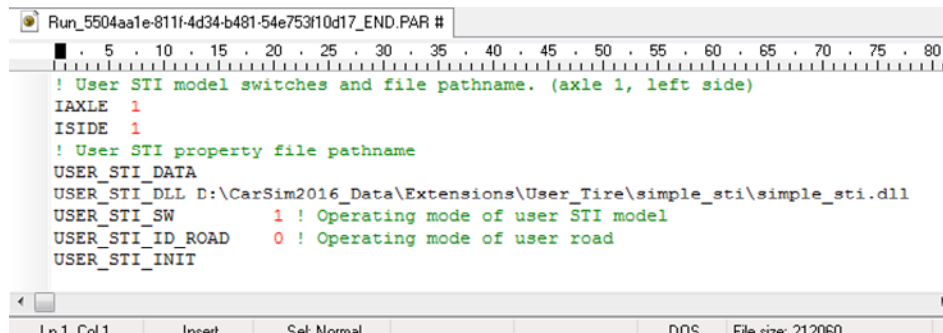


*Listing 4. Example source code for writing to an echo file.*

```
// Use function from imap.c to echo iaxle, iside, iunit
if (sDualsArePossible) {
    gPrintEcho (ifile, "\n! User STI model switches and file pathname. (%s)\n",
        sTab_label_tire(itire));
}
else {
    gPrintEcho (ifile, "\n! User STI model switches and file pathname. (%s)\n",
        sTab_label_wheel(itire));
}

if (sDualsArePossible) sEcho_tire (ifile, itire);
else sEcho_wheel (ifile, itire);

gPrintEcho (ifile, "! User STI property file pathname\n");
gWrite_echo_line (ifile, "USER_STI_DATA", mp->chtdst, "");
gWrite_echo_line (ifile, "USER_STI_DLL", mp->chsdst, "");
gWritei (ifile, "USER_STI_SW", mp->iswtch,
    "! Operating mode of user STI model");
gWritei (ifile, "USER_STI_ID_ROAD", mp->idroad,
    "! Operating mode of user road ");
if (strcmp(mp->chrdst, "")) {
    gPrintEcho (ifile, "! User STI road data file pathname (optional)\n");
    gWrite_echo_line (ifile, "USER_STI_ROAD", mp->chrdst, "");
}
gPrintEcho(ifile, "USER_STI_INIT\n");
```



```
Run_5504aa1e-811f-4d34-b481-54e753f10d17_END.PAR #
! User STI model switches and file pathname. (axle 1, left side)
IAXLE 1
ISIDE 1
! User STI property file pathname
USER_STI_DATA
USER_STI_DLL D:\CarSim2016_Data\Extensions\User_Tire\simple_sti\simple_sti.dll
USER_STI_SW 1 ! Operating mode of user STI model
USER_STI_ID_ROAD 0 ! Operating mode of user road
USER_STI_INIT
```

*Figure 15. External tire data parameters displayed on echo files*

In this example code, `gPrintEcho` is used to write text on the file associated with the pointer `ifile`. `gPrintEcho` and `gWrite_echo_line` are the callback functions that make use of VS API functions from within the VS Solver. They are described later (page 20). `sTab_labe_tire` function converts the tire number to a character string of the tire location (e.g. axle 1, left side). All functions whose name starts with “s” (e.g. `sEcho_wheel`) are local functions.

### *vs\_sti\_SetVisible*

Declaration:

```
void vs_sti_SetVisible(void);
```

Set the visibility of the external tire data parameters. This is used in case the VS STI Module has parameters that are not always applicable. This is an opportunity for the module to mark unused parameters so they will not be written to the Echo file.

### *vs\_sti\_calc*

Declaration:

```
void vs_sti_calc(int itire, vs_real time,  
    vs_real xw0, vs_real yw0, vs_real zw0,  
    vs_real yaw, vs_real pitch, vs_real roll,  
    vs_real vwx, vs_real vwy, vs_real vwz,  
    vs_real avwx, vs_real avwy, vs_real avwz,  
    vs_real rot, vs_real w,  
    vs_real *fxw0, vs_real *fyw0, vs_real *fzw0,  
    vs_real *mxw0, vs_real *myw0, vs_real *mzw0,  
    vs_real **infVar);
```

VS Solvers call this function to calculate the tire forces and moments, based on kinematical variables at each wheel center such as position coordinates ( $xw0$ ,  $yw0$  and  $zw0$ ), velocities ( $vwx$ ,  $vwy$  and  $vwz$ ), orientation angles ( $yaw$ ,  $pitch$  and  $roll$ ), angular speeds ( $avwx$ ,  $avwy$  and  $avwz$ ), wheel rotation angle and speed ( $rot$  and  $w$ ).

This function returns three forces ( $fxw0$ ,  $fyw0$  and  $fzw0$ ) and three moments ( $mxw0$ ,  $myw0$  and  $mzw0$ ) that are applied at the wheel center of the vehicle model. Definition of the variables are summarized below. The descriptions are based on the axis systems and coordinate systems described in separate document, [Vehicle System Terminology](#).

Position coordinates ( $xw0$ ,  $yw0$  and  $zw0$ ):  $\mathbf{X}_E$ ,  $\mathbf{Y}_E$ , and  $\mathbf{Z}_E$  coordinates of the wheel center with respect to the earth-fixed coordinate system origin.

Velocities ( $vwx$ ,  $vwy$  and  $vwz$ ): wheel center velocities in the wheel axis system ( $\mathbf{X}_W$ ,  $\mathbf{Y}_W$ ,  $\mathbf{Z}_W$ ) to the earth-fixed axis system.

Orientation angles ( $yaw$ ,  $pitch$  and  $roll$ ): Euler angles (the rotation order is yaw, pitch and roll) transforming from earth-fixed coordinate system ( $\mathbf{X}_E$ ,  $\mathbf{Y}_E$ ,  $\mathbf{Z}_E$ ) to wheel axis coordinate system ( $\mathbf{X}_W$ ,  $\mathbf{Y}_W$ ,  $\mathbf{Z}_W$ ).

Angular speeds ( $avwx$ ,  $avwy$  and  $avwz$ ): rotation speeds of the wheel carrier to the earth-fixed axis system expressed in the wheel axis system ( $\mathbf{X}_W$ ,  $\mathbf{Y}_W$ ,  $\mathbf{Z}_W$ ).

Wheel rotation angle and speed ( $rot$  and  $w$ ): rotation angle and angular speed of spinning wheel with respect to the wheel carrier.

Forces ( $fxw0$ ,  $fyw0$  and  $fzw0$ ): forces transformed from the tire forces at CTC to wheel center expressed in the wheel axis system ( $\mathbf{X}_W$ ,  $\mathbf{Y}_W$ ,  $\mathbf{Z}_W$ ).

Moments ( $mxw0$ ,  $myw0$  and  $mzw0$ ): moments transformed from the tire forces and moments at CTC to wheel center expressed in the wheel axis system ( $\mathbf{X}_W$ ,  $\mathbf{Y}_W$ ,  $\mathbf{Z}_W$ ).

The VS Solver calls `vs_sti_calc` for each tire using the tire identification number `itire`. For example, if all four tires are specified as external tires, VS Solver calls `vs_sti_calc` four times with `itire` of 1 to 4.

`**infVar` is pointer to the output array from the external tire model. The output values from the external tire model are fed to overwrite the output variables of the VS Solvers. In version 2016, the way to overwrite the output variables are hard-coded as shown in Table 4.

Table 4. VS output variables overwritten from the external tire model outputs.

Output variables in the VS Solver	Array element from the external tire model
Fx	<code>infVar[0]</code>
Fy	<code>infVar[1]</code>
Fz	<code>infVar[2]</code>
mx	<code>infVar[3]</code>
my	<code>infVar[4]</code>
mz	<code>infVar[5]</code>
gamma	<code>infVar[8]</code>
alpha	<code>infVar[50]</code>
kappa	<code>infVar[51]</code>
mux	<code>infVar[16]</code>
muy	<code>infVar[17]</code>
CompT	<code>infVar[43]</code>
vx	<code>infVar[45]</code>
vy	<code>infVar[46]</code>
Xctc	<code>infVar[65]</code>
Yctc	<code>infVar[66]</code>
Zgnd	<code>infVar[67]</code>
dzdx	<code>-infVar[68]/infVar[70]</code>
dzdy	<code>-infVar[69]/infVar[70]</code>

**Note** The way to overwrite the VS output variables from the output array of the external model is based on an earlier custom model. We plan to make the mapping customizable in future versions.

## `vs_sti_calc2`

Declaration:

```
void vs_sti_calc2(int iwheel, vs_real ldual, vs_real time,
vs_real xw0, vs_real yw0, vs_real zw0,
vs_real yaw, vs_real pitch, vs_real roll,
vs_real vwx, vs_real vwy, vs_real vwz,
vs_real avwx, vs_real avwy, vs_real avwz,
vs_real rot, vs_real w,
vs_real *fxw0, vs_real *fyw0, vs_real *fzw0,
vs_real *mxw0, vs_real *myw0, vs_real *mzw0,
vs_real **infVar, vs_real **infVar2);
```

TruckSim has a capability to define dual-tires which use two tires on each side of axle. As shown in Figure 16, if you check the box for Dual tires (1) specifying the spacing (2) between the tires on **Tires: Vehicle Unit with 2 Axles (3 Axles or 4 Axles)** screen, the VS Solvers call function `vs_sti_calc2` to calculate the total forces and moments from the dual tires at the center between the two tires on the axle.

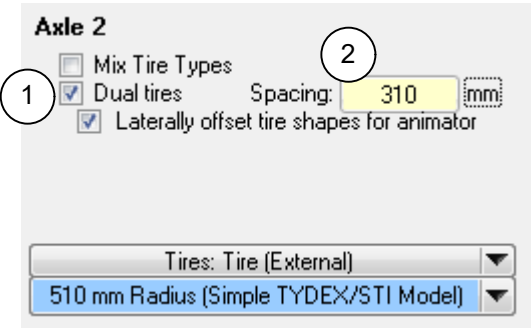


Figure 16. Selecting dual tires on TruckSim Tires screen.

The kinematical values passed from the VS Solvers are all referenced at the center point between the dual tires on the axle. Definitions of the variables are based on the same axis systems and coordinate systems as `vs_sti_calc`.

`iwheel` is the corner identification number (not the tire identification number `itire`) that normally axle 1, left is 1; right is 2; axle 2, left is 3; right is 4.

***vs\_sti\_exit***

Declaration:

```
void vs_sti_exit (int i);
```

VS Solvers call this function at the end of simulation for the number of tires with incrementing `i` starting from 0. For example, four-wheeled VS Solvers call this function four times with `i` ranging from 0 to 3.

**Callbacks from VS Solvers**

VS Solvers include many functions that may be made available for use in the VS STI module as callbacks. For example, Listing 5 shows code for the VS STI function `vs_sti_write_to_logfile` that makes used of a VS function `vs_write_to_logfile`. A pointer to the VS function is assigned to a local function `gWrite_to_logfile`. The enables the local function `gWrite_to_logfile` to apply the standard write function from the VS Solver.

Table 5 lists the callback types from the original VS functions and their installer functions.

<b>Note</b>	All installer functions listed in Table 5 are called from the VS Solvers.
	Therefore, all of them must exist in the VS STI module, even the functions have no internal code.

Listing 5. Source code for `vs_sti_calc` in example.

```
VS_STI_WRITE_TO_LOGFILE_FUNC gWrite_to_logfile;
VS_API_EXPORT void vs_sti_write_to_logfile(VS_STI_WRITE_TO_LOGFILE_FUNC function_name)
{
    gWrite_to_logfile = function_name;
}
```

Table 5. Callback types from VS functions, their installers and original VS API functions.

Callback Type and Installer Functions	Original VS functions
<code>gWrite_to_logfile</code> <code>vs_sti_write_to_logfile</code>	<code>vs_write_to_logfile</code>
<code>gGet_var_ptr_int</code> <code>vs_sti_get_var_ptr_int</code>	<code>vs_get_var_ptr_int</code>
<code>gGet_var_ptr</code> <code>vs_sti_get_var_ptr</code>	<code>vs_get_var_ptr</code>
<code>gWritei</code> <code>vs_sti_writei</code>	<code>vs_write_i_to_echo_file</code>
<code>gWrite_echo_line</code> <code>vs_sti_write_echo_line</code>	<code>vs_write_echo_line</code>
<code>gSet_sym_attribute</code> <code>vs_sti_set_sym_attribute</code>	<code>vs_set_sym_attribute</code>
<code>gPrintf_error</code> <code>vs_sti_printf_error</code>	<code>vs_printf_error</code>
<code>gInstall_free_function</code> <code>vs_sti_install_free_function</code>	<code>vs_install_free_function</code>
<code>gDefine_sv</code> <code>vs_sti_define_sv</code>	<code>vs_define_sv</code>
<code>gDefine_out</code> <code>vs_sti_define_out</code>	<code>vs_define_out</code>
<code>gHave_keyword_in_database</code> <code>vs_sti_have_keyword_in_database</code>	<code>vs_have_keyword_in_database</code>
<code>gGet_filebase_name</code> <code>vs_sti_get_filebase_name</code>	<code>vs_get_filebase_name</code>
<code>gTyparr</code> <code>vs_sti_typarr</code>	<code>vscStiTyparr</code> (not exported for API)
<code>gGetAllTiresAlike</code> <code>vs_sti_GetAllTiresAlike</code>	<code>vscGetAllTiresAlike</code> (not exported for API)
<code>gSetTyparrVisible</code> <code>vs_sti_set_typarr_visible</code>	<code>vscSetStiTyparrVisible</code> (not exported for API)
<code>gTiresSetPars</code> <code>vs_sti_TiresSetPars</code>	<code>vscSetTiresSetPars</code> (not exported for API)
<code>gPrintEcho</code> <code>vs_sti_print_echo</code>	<code>fprintf</code> (standard C function)
<code>gGet_road_contact_ext</code> <code>vs_sti_get_road_contact_ext</code>	<code>vsc_get_road_contact</code>
<code>gDatabaseDir</code> <code>vs_sti_DatabaseDir</code>	<code>vspDatabaseDir</code> (not exported for API)

Most of the VS functions listed in the table are VS API functions that are documented in [The VehicleSim API](#) reference manual. The callback functions that are not part of the VS API are described below.

### *gTyparr*

Declaration:

```
vs_real gTyparr(int itire, int x);
```

Return:

Tire parameter (TYPARR) value

This function accesses the tire parameter table data (TYPARR) (root keyword: STI\_TYPARR) passing the parameter number (first column of the table starting from 0) returning the parameter value (second column of the table).

### *gGetAllTiresAlike*

Declaration:

```
int gGetAllTiresAlike (void);
```

Return:

1: if specified for all tires are the same, 0: for individual settings

### *gTiresSetPars*

Declaration:

```
void gTiresSetPars (int itire, vs_real rre, vs_real r_free,  
vs_real kt_ext, vs_real kt2_ext,  
vs_real iyy, vs_real ms);
```

Use this function to pass the effective rolling radius *rre*, free rolling radius *r\_free*, and vertical stiffness *kt\_ext* of the external tire model for the tire identified by *itire* to make a correct initialization of the vehicle height and correct calculations for the wheel velocities.

*Kt2\_ext*, *iyy* and *ms* are used by other 3<sup>rd</sup>-party tire models and they are no effect for the user defined external tire models.

### *gSetTyparrVisible*

Declaration:

```
void gSetTyparrVisible (int itire, int x);
```

Set the tire parameter table data (TYPARR) (root keyword: STI\_TYPARR) written in echo files or not. If *x* is passed as 1, the TYPARR table for the tire identified by *itire* is written in echo files. If the external tire model reads a tire property data file rather than the table data in VS format, pass *x* as 0 to not echo the TYPARR table.

### *gDatabaseDir*

Declaration:

```
char * gDatabaseDir (void);
```

Return:

Path and folder name of the database directory (e.g. C:\CarSim\_Data).

## References

1. TYDEX – Working Group on Tyre Modelling and Description, “STI – Standardized Interface Tyre Model – Vehicle Model (Release 1.4)”, December 1996.