Automating Runs with the VS API

VS Solvers are libraries that support the building and running of VS Math Models for CarSim, TruckSim, BikeSim, and SuspensionSim. The Windows versions of these products include both 32- and 64-bit solvers, e.g., `carsim_32.dll` and `carsim_64.dll` for CarSim. For Linux, they are 64-bit .so library files.

On Windows, the 32-bit VS Solver DLL files are normally used from the main application program in the products, e.g., `carsim.exe` for CarSim. These applications, called *VS Browsers*, provide a GUI for interacting with the models and managing a database of files that are mostly read as input by the VS Math Model, or written as output by the model. Alternatively, a VS Solver may be loaded and controlled by other Windows programs.

Programs that are used mainly to load a DLL and use its functions are sometimes called *wrappers*.

In support of automation work in which a VS Browser is not needed, the Windows versions of the products include the wrappers `VS_SolverWrapper_CLI_32.exe` and `VS_SolverWrapper_CLI_64.exe`. These are described in the Technical Memo *VS Solver Wrapper*.

There may be scenarios in which a user-defined custom wrapper is needed or preferred. This document describes examples of custom wrappers that are provided in SDK CPAR archives for use with MATLAB, C/C++, Visual Basic (VB), and Python. The remainder of this document shows how to build and run a VS Math Model from a program written with MATLAB, C/C++, VB, and Python

Another Technical Memo, *Extending VS Math Models with VS Commands and the VS API*, extends these methods to show how VS Math Models in CarSim, TruckSim, and BikeSim are extended with custom wrappers written in MATLAB and C/C++.

The comprehensive reference manual for VS Math Models is the *VS Math Models Reference Manual*; the reference manual for VS application program interface (API) is *The VehicleSim API: Accessing and Extending VS Math Models*.

Building and Running a VS Math Model: Overview

When running on Windows OS, the VS Solver is a DLL file. The top-level input file for the VS Solver is a simulation control file that in turn contains pathnames for various files of interest, including the pathname for the solver DLL. This file is called the *Simfile* and has the default name `simfile.sim`. Details of Simfile contents are provided the *VS Math Models Reference Manual*.

The normal way for a wrapper program to build and run a VS Math Model on Windows is:

1. Read the Simfile to get the pathname for the DLL file.
2. Load the VS Solver DLL.
3. Activate API functions from the DLL.
4. Use the VS API function `vs_run` to build the model and make the run. This function also requires the name of the Simfile, to obtain more details of input and output file names.
5. Unload the DLL.

Location of Examples

The VehicleSim API and its examples can be found within the VS SDK category, available in an SDK CPAR archive for each product. The VS SDK is a collection of tools, libraries, and files that are bundled together to provide a development framework for interacting with the VehicleSim family of products. The VS SDK can be downloaded from the Mechanical Simulation website.

The CarSim examples referenced in this document are contained in datasets from the CarSim SDK CPAR archive file. Comparable examples for BikeSim and TruckSim are also available in CPAR archive files.

The Stand-alone License Manager Control Option

A VS Solver DLL needs license information to run. When run under the control of a VS Browser, the VS Browser manages the license information automatically.

When a VS Solver DLL is used from software other than the VS Browser, one way to provide the necessary license information is to have the VS Browser running (it may be minimized) while you work in the environment of the external software. This method is convenient when you are making changes to vehicle datasets or test procedures from within the browser.

A second way to provide license information is with the stand-alone License Manager program provided with all VehicleSim products. The program file is `BSLM.exe`, `CSLM.exe`, `TSLM.exe`, or `SSLM.exe`, located in either the `_Prog\Programs` folder (e.g., `CarSim_Prog \Programs`) or within the Utilities folder of the VS SDK. This program provides the same license control that is ordinarily provided by the VS Browser. To use it, launch the `BSLM/CSLM/TSLM/SSLM .exe file` (e.g., double click on the file name) before you start using any software that makes use of VS Solver DLLs. You can also make a shortcut on your desktop or in another convenient location, or add it to your start menu so it is always active after you log in. Details about the license manager are provided in a separate tech memo *The VehicleSim*

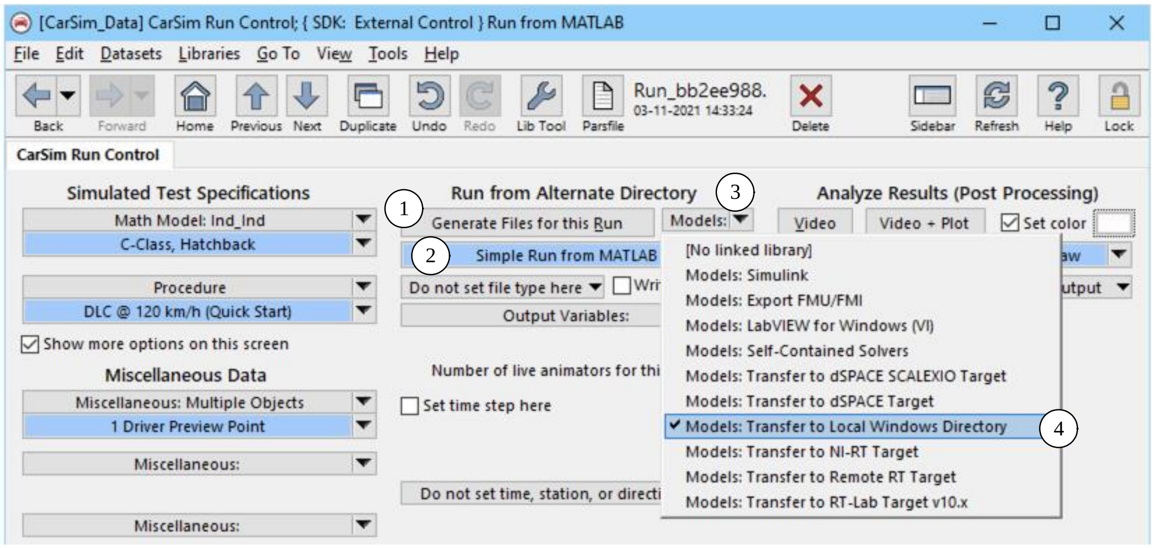
License Manager Utility, covering topics such as changing the settings and running older versions of your software.

Using the License Manager application is convenient when you are using a custom interface program to produce the input files for the VS Solver, or when many runs are made in an automated environment, such as optimization. VSLM is a small program requiring minimal resources, so your system performance will not be affected by having it run in the background.

Running a VS Math Model from MATLAB

Building and running a VS Math Model from MATLAB requires statements in MATLAB to read a Simfile, load the VS Solver DLL, access its functions, and apply the API function `vs_run`.

If you have set up the run from the VS Browser, you typically want the Simfile to be generated automatically in order to use the proper datasets for the VS Math Model. To do this, make a dataset in the **Run Control** library with settings for the run (vehicle, procedure, etc.), and use the **Models** drop-down control ③ (Figure 1) to link to the library **Models: Transfer to Local Windows Directory** ④. Make a link ② to a dataset in the linked library.



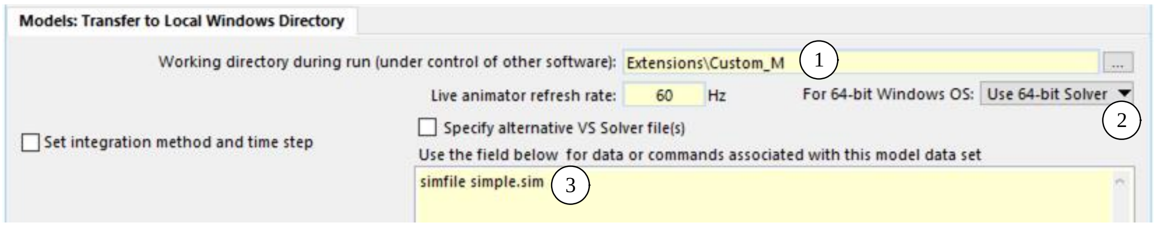
[CarSim_Data] CarSim Run Control; { SDK: External Control } Run from MATLAB
File Edit Datasets Libraries Go To View Tools Help
Back Forward Home Previous Next Duplicate Undo Redo Lib Tool Parsfile
Run_bb2ee988.
03-11-2021 14:33:24
Delete Sidebar Refresh Help Lock
CarSim Run Control
Simulated Test Specifications
Math Model: Ind_Ind
C-Class, Hatchback
Procedure
DLC @ 120 km/h (Quick Start)
Show more options on this screen
Miscellaneous Data
Miscellaneous: Multiple Objects
1 Driver Preview Point
Miscellaneous:
Miscellaneous:
Run from Alternate Directory
1
Generate Files for this Run
2
Simple Run from MATLAB
Do not set file type here
Wri
Output Variables:
Number of live animators for thi
Set time step here
Do not set time, station, or directi
3
Models:
[No linked library]
Models: Simulink
Models: Export FMU/FMI
Models: LabVIEW for Windows (VI)
Models: Self-Contained Solvers
Models: Transfer to dSPACE SCALEXIO Target
Models: Transfer to dSPACE Target
✓ Models: Transfer to Local Windows Directory
4
Models: Transfer to NI-RT Target
Models: Transfer to Remote RT Target
Models: Transfer to RT-Lab Target v10.x
Analyze Results (Post Processing)
Video
Video + Plot
Set color
aw
utput

Figure 1. Link to the library Models: Transfer to Local Windows Directory.

Use the linked dataset to identify the folder containing the MATLAB file ① (Figure 2). In the same dataset, use the drop-down control for choosing a 32/64 bit solver ② to match your version of MATLAB. This will cause the pathname for the appropriate VS Solver to be written in the Simfile.

The dataset that specifies the directory may also be used to specify a custom Simfile name with the keyword `simfile` ③. For the simple run, the Simfile is named `simple.sim`.

Return to the **Run Control** screen (Figure 1) and click the button **Generate Files for this Run** ① to generate the Simfile in the specified working directory, along with a matching `All.Par` file with all settings from the database that will be used by the VS Math Model for the run.


Models: Transfer to Local Windows Directory

Working directory during run (under control of other software): Extensions\Custom_M ① ...

Live animator refresh rate: 60 Hz

For 64-bit Windows OS: Use 64-bit Solver ②

☐ Set integration method and time step

☐ Specify alternative VS Solver file(s)

Use the field below for data or commands associated with this model data set

simfile simple.sim ③

Figure 2. Specify the folder containing MATLAB files.

To run the simulation from MATLAB, load the M file (Figure 3) and run it using a menu control, MATLAB command, or click a Run button (the details for how you control MATLAB depend on the version). With the contents of the M file displayed (`Simple.m` ③), use the **EDITOR** tab ① to view the **Run** button ②. Click the **Run** button to execute the code in the M file, with results shown in Command Window ⑥. When the run is complete, plots and animations can be viewed from the VS Browser, from the Run Control screen (Figure 1).

This example performs the following steps:

1. It clears the command window and checks to ensure a DLL is not already loaded (lines 7 -10).
2. It sets the name of a variable `simfile` to the string '`simple.sim`' and uses an M-file function `vs_dll_path` (④ and ⑤) to scan the specified Simfile for the location of the VS Solver DLL, and assign that pathname to the variable `SolverPath` (lines 14 and 18).
3. The MATLAB function `loadlibrary` loads the DLL along with a header file (`vs_api_def_m.h`) that gives access to VS API functions (lines 21 – 23).
4. The simulation is run with the VS API function `vs_run` and the same `simfile` variable (line 26).
5. The DLL is unloaded (line 30).

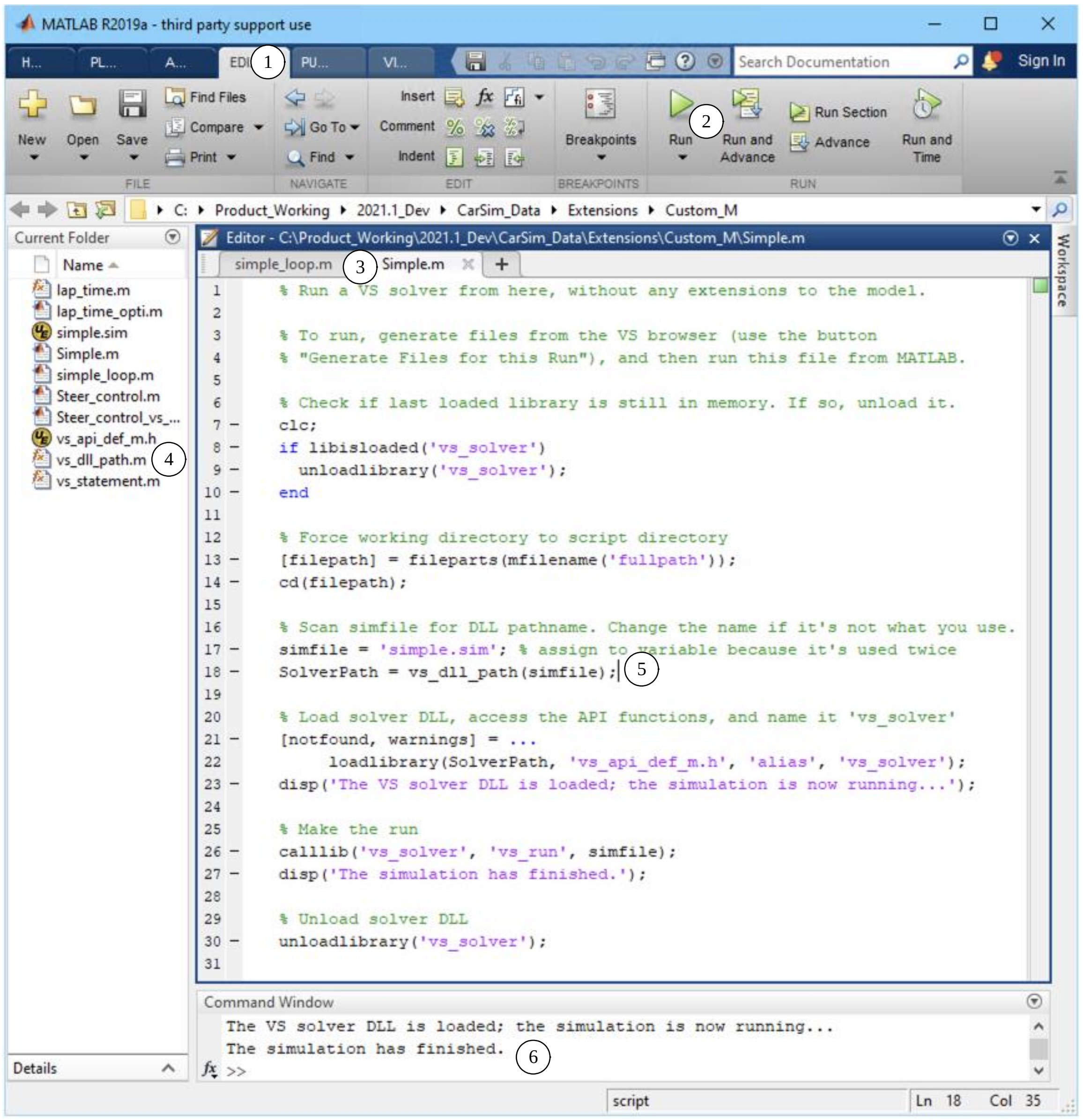
The status of the run is printed in the Command window ⑥.

Running a VS Math Model from ANSI C

Figure 4 shows the source code for an ANSI C wrapper program (`solver_simple.c`) that reads a simulation control file, gets the name of the VS Solver library, and makes the run. The EXE is built from this file and another (`vs_get_api.c`) that provides access to the API functions in the VS Solver.

In the example, the name of the Simfile is an optional argument (lines 16 – 22) that defaults to `"simfile.sim"` if not provided (line 19). The Simfile is scanned to determine the pathname of the DLL with the function `vs_get_dll_path` (line 23). This function (from vs`_get_api.c`) looks for an explicit pathname; if not found, it is generated from the “bitness” (32 of 64) of the wrapper along with the product name (from the Simfile). When successful, the DLL is loaded (line 24). API functions are accessed with the function `vs_get_api` (line 27).

Two API functions are used in this example: `vs_run` (line 30) and `vs_get_error_message` (lines 32 and 34).


MATLAB R2019a - third party support use
EDI 1
Run 2
Simple.m 3
vs_dll_path.m 4
5
6
Current Folder
Editor - C:\Product_Working\2021.1_Dev\CarSim_Data\Extensions\Custom_M\Simple.m
Command Window

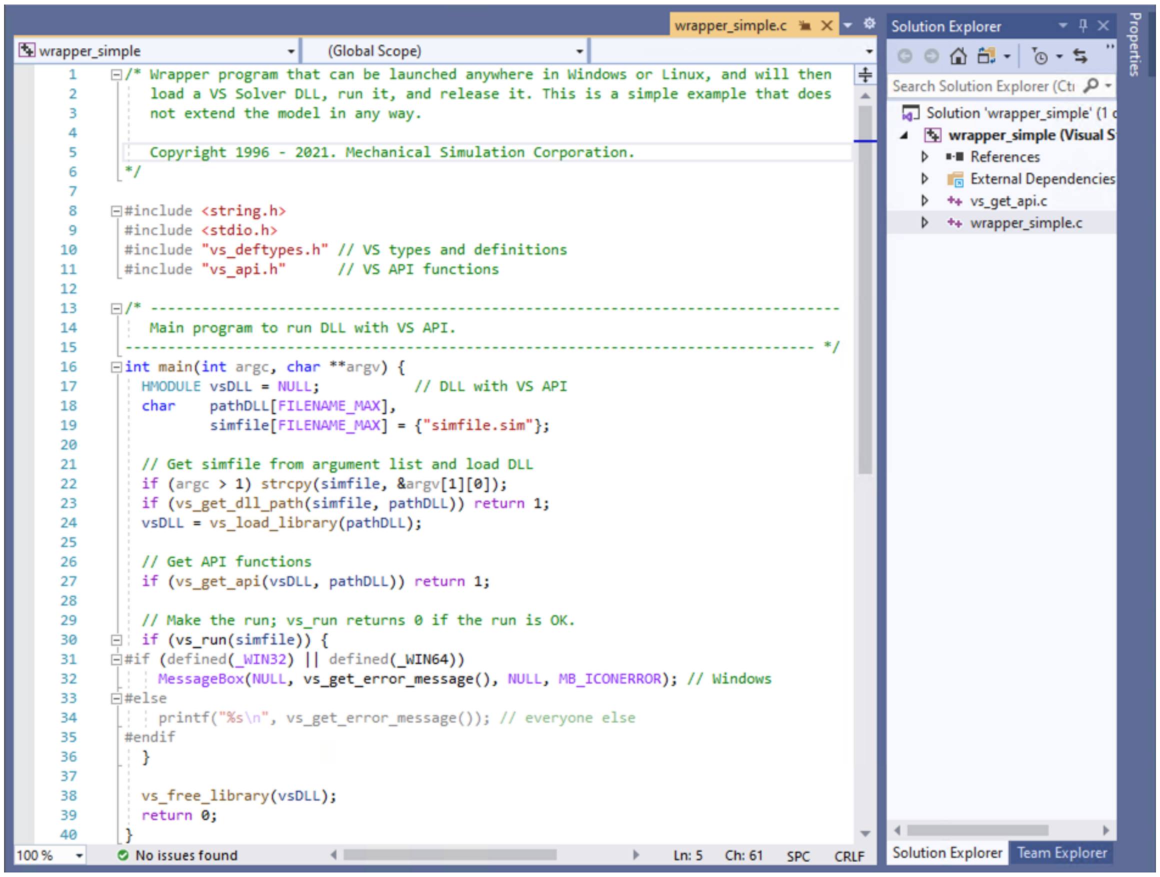
```
% Run a VS solver from here, without any extensions to the model.

% To run, generate files from the VS browser (use the button
% "Generate Files for this Run"), and then run this file from MATLAB.

% Check if last loaded library is still in memory. If so, unload it.
clc;
if libisloaded('vs_solver')
  unloadlibrary('vs_solver');
end

% Force working directory to script directory
[filepath] = fileparts(mfilename('fullpath'));
cd(filepath);

% Scan simfile for DLL pathname. Change the name if it's not what you use.
simfile = 'simple.sim'; % assign to variable because it's used twice
SolverPath = vs_dll_path(simfile);

% Load solver DLL, access the API functions, and name it 'vs_solver'
[notfound, warnings] = ...
     loadlibrary(SolverPath, 'vs_api_def_m.h', 'alias', 'vs_solver');
disp('The VS solver DLL is loaded; the simulation is now running...');

% Make the run
calllib('vs_solver', 'vs_run', simfile);
disp('The simulation has finished.');

% Unload solver DLL
unloadlibrary('vs_solver');
```

```
The VS solver DLL is loaded; the simulation is now running...
The simulation has finished.
>>
```

Figure 3. Loading and running a VS Solver from MATLAB.

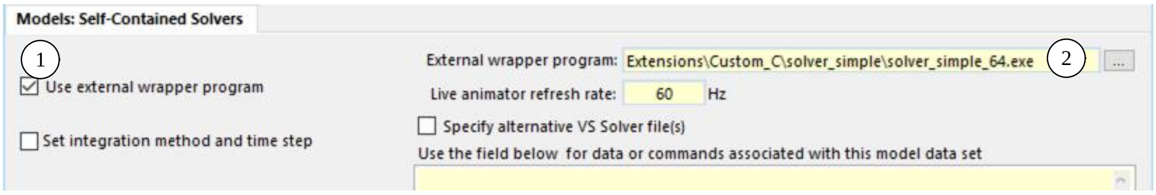
Use the **Models: Self-Contained Solvers** library to make a dataset for the wrapper (Figure 5). Check the box to use an external wrapper program ① and specify the pathname for the compiled wrapper program ②. As noted above, the example wrapper will use the VS Solver with the same bitness as the wrapper. In this example, the wrapper name has the bitness in the name: 64 ②.

To run a simulation with this wrapper, make a dataset in the **Run Control** screen with settings for the run (vehicle, procedure, etc.). Use the **Models** drop-down control ③ (Figure 6) to link to the library **Models: Self-Contained Solvers** ④. Make a link to the appropriate **Models** dataset in the linked library ②.



```c
/* Wrapper program that can be launched anywhere in Windows or Linux, and will then
   load a VS Solver DLL, run it, and release it. This is a simple example that does
   not extend the model in any way.

   Copyright 1996 - 2021. Mechanical Simulation Corporation.
*/

#include <string.h>
#include <stdio.h>
#include "vs_deftypes.h" // VS types and definitions
#include "vs_api.h"      // VS API functions

/* ----------------------------------------------------------------------------
   Main program to run DLL with VS API.
------------------------------------------------------------------------------ */
int main(int argc, char **argv) {
  HMODULE vsDLL = NULL;           // DLL with VS API
  char    pathDLL[FILENAME_MAX],
          simfile[FILENAME_MAX] = {"simfile.sim"};

  // Get simfile from argument list and load DLL
  if (argc > 1) strcpy(simfile, &argv[1][0]);
  if (vs_get_dll_path(simfile, pathDLL)) return 1;
  vsDLL = vs_load_library(pathDLL);

  // Get API functions
  if (vs_get_api(vsDLL, pathDLL)) return 1;

  // Make the run; vs_run returns 0 if the run is OK.
  if (vs_run(simfile)) {
#if (defined(_WIN32) || defined(_WIN64))
    MessageBox(NULL, vs_get_error_message(), NULL, MB_ICONERROR); // Windows
#else
    printf("%s\n", vs_get_error_message()); // everyone else
#endif
  }

  vs_free_library(vsDLL);
  return 0;
}
```

Figure 4. Loading the DLL and making a run from C.



Models: Self-Contained Solvers

☑ Use external wrapper program

☐ Set integration method and time step

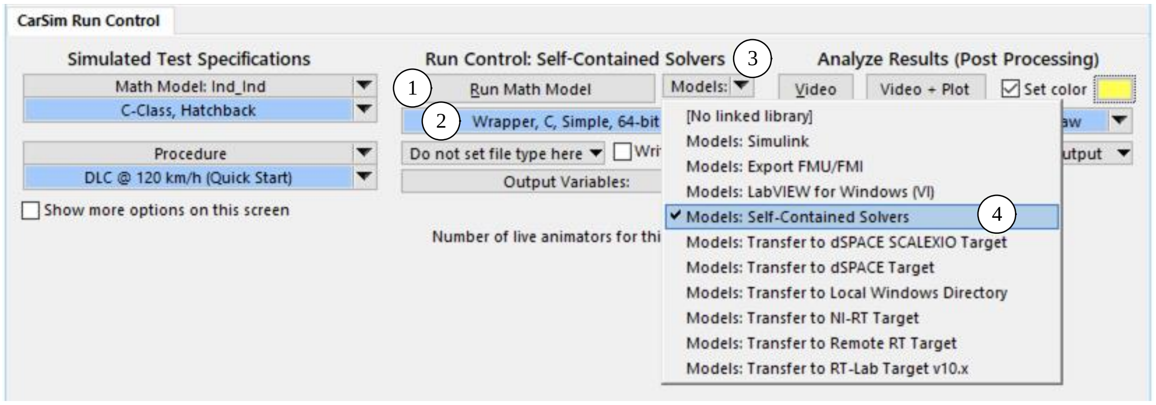
External wrapper program: Extensions\Custom_C\solver_simple\solver_simple_64.exe

Live animator refresh rate: 60 Hz

☐ Specify alternative VS Solver file(s)

Use the field below for data or commands associated with this model data set

Figure 5. Identify the pathname to the compiled EXE wrapper program.



CarSim Run Control

Simulated Test Specifications

Math Model: Ind_Ind

C-Class, Hatchback

Procedure

DLC @ 120 km/h (Quick Start)

☐ Show more options on this screen

Run Control: Self-Contained Solvers

Wrapper, C, Simple, 64-bit

Do not set file type here

Output Variables:

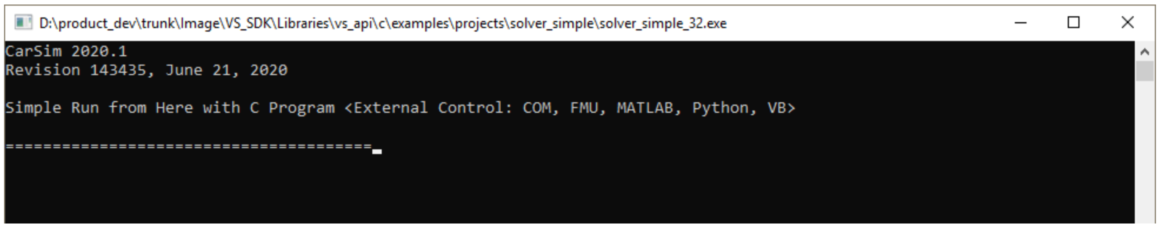
Analyze Results (Post Processing)

☑ Set color

[No linked library]
Models: Simulink
Models: Export FMU/FMI
Models: LabVIEW for Windows (VI)
✓ Models: Self-Contained Solvers
Models: Transfer to dSPACE SCALEXIO Target
Models: Transfer to dSPACE Target
Models: Transfer to Local Windows Directory
Models: Transfer to NI-RT Target
Models: Transfer to Remote RT Target
Models: Transfer to RT-Lab Target v10.x

Figure 6. Link to the library Models: Self-Contained Solvers.

Run the wrapper program by clicking the **Run Math Model** button 1. The wrapper program will run as a console application (Figure 7). When operated in this manner, the behavior of the VS Math Model is the same as if the simulation were run from the Browser. Results are visualized using the same tools, Echo files have the same appearance, and simulation results are identical to those obtained without the wrapper program.


D:\product_dev\trunk\Image\VS_SDK\Libraries\vs_api\c\examples\projects\solver_simple\solver_simple_32.exe

```
CarSim 2020.1
Revision 143435, June 21, 2020

Simple Run from Here with C Program <External Control: COM, FMU, MATLAB, Python, VB>

======================================
```

Figure 7. EXE wrapper program running as a console application.

This EXE program can be used from other environments that are not able to easily load and run DLLs directly. Because it can be given the Simfile name as an optional argument, it is not always necessary to write the Simfile in the same folder as the EXE. That is, the EXE can be deployed with a Windows run command that specifies a simulation control file with any name, in any valid Windows location. The only requirement is that the VS Browser or the VSLM utility program must be running to provide license support.

Note As mentioned earlier, VS products include simple wrapper EXE programs that support a command-line interface: `VS_SolverWrapper_CLI_32.exe` and `VS_SolverWrapper_CLI_32.exe`. These exist to support automation without requiring users to create custom wrappers.

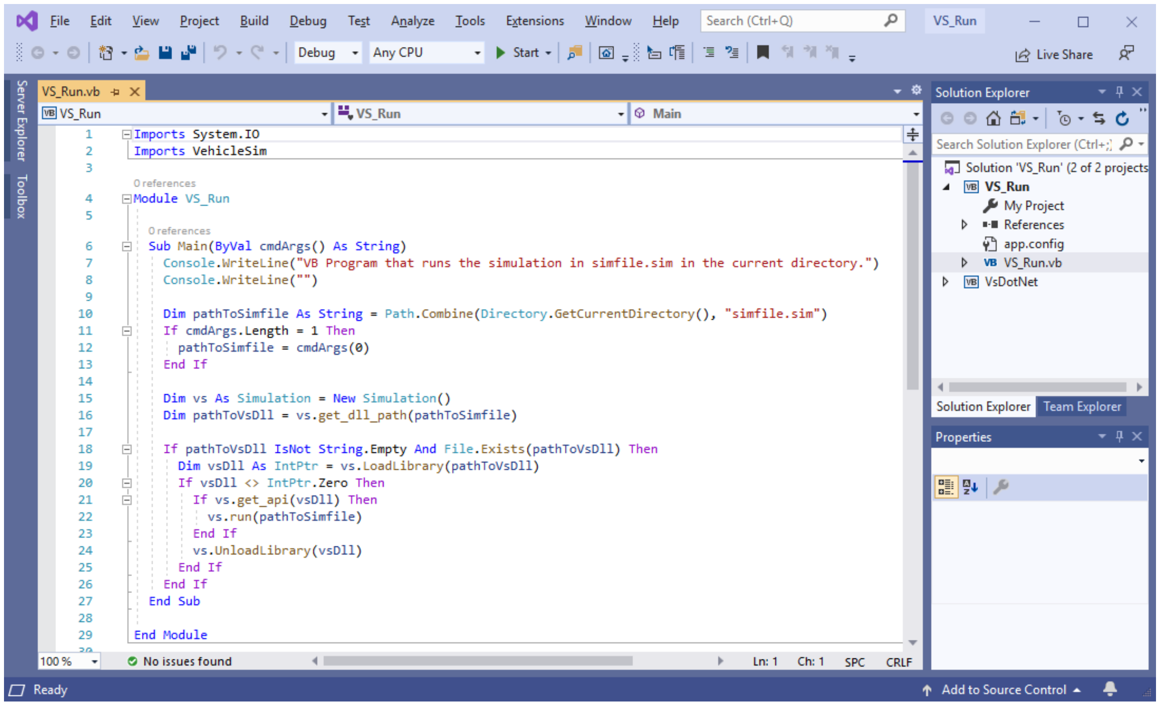
Running a VS Math Model from Visual Basic (VB)

A wrapper program can be written in Visual Basic (VB) to load the VS Solver and run a VS Math Model.

A complication is that VB uses a different internal representation for strings (e.g., the pathname of the Simfile). VehicleSim Products include a VisualBasic.NET project called `VsDotNet`. This project contains a `Simulation` class (see `Simulation.vb`) that provides a .NET wrapper for some of the VS API functions, such as the handling of string variables.

The example code in `VS_Run.vb` (Figure 8) runs a simulation using the same general method shown for MATLAB and C.

1. The name of a Simfile (the default is "`simfile.sim`") is assigned to a string variable `pathToSimfile` (lines 10 - 13).
2. An instance of the `Simulation` class is assigned to the variable `vs` (line 15), and a pathname to the DLL file is obtained and assigned to the string variable `PathToVsDll` by scanning the specified Simfile (line 16).



```
 1  Imports System.IO
 2  Imports VehicleSim
 3
    0 references
 4  Module VS_Run
 5
      0 references
 6    Sub Main(ByVal cmdArgs() As String)
 7      Console.WriteLine("VB Program that runs the simulation in simfile.sim in the current directory.")
 8      Console.WriteLine("")
 9
10      Dim pathToSimfile As String = Path.Combine(Directory.GetCurrentDirectory(), "simfile.sim")
11      If cmdArgs.Length = 1 Then
12        pathToSimfile = cmdArgs(0)
13      End If
14
15      Dim vs As Simulation = New Simulation()
16      Dim pathToVsDll = vs.get_dll_path(pathToSimfile)
17
18      If pathToVsDll IsNot String.Empty And File.Exists(pathToVsDll) Then
19        Dim vsDll As IntPtr = vs.LoadLibrary(pathToVsDll)
20        If vsDll <> IntPtr.Zero Then
21          If vs.get_api(vsDll) Then
22            vs.run(pathToSimfile)
23          End If
24          vs.UnloadLibrary(vsDll)
25        End If
26      End If
27    End Sub
28
29  End Module
```

Figure 8. VB program that loads a VS Solver DLL and makes a run.

3. If a file named `PathToVsDll` exists (line 18), then the DLL is loaded (line 19) and some of the API functions are accessed (line 21).
4. The API function `vs_run` is applied using the `Simulation` function `vs.run` (line 22).
5. The DLL is unloaded (line 24).

EXE programs such as this can be run from the **Run Control** screen (as shown before for the C example), or from an environment outside the VS database (as shown before for the MATLAB example).

To run from the VS Browser, link to a dataset in the **Models: Self-Contained Solvers** library (Figure 6), check the box to use an external wrapper program (1) (Figure 5) and specify the pathname for the compiled wrapper program (2).

To run from the MSVS environment, link to a dataset from the library **Models: Transfer to Local Windows Directory** (4) (Figure 1, page 3). In the linked dataset (Figure 2, page 4), specify the directory in which the EXE file will be located (1). If you are not using a default name for the Simfile, specify a name, as was done earlier for the MATLAB example (3).

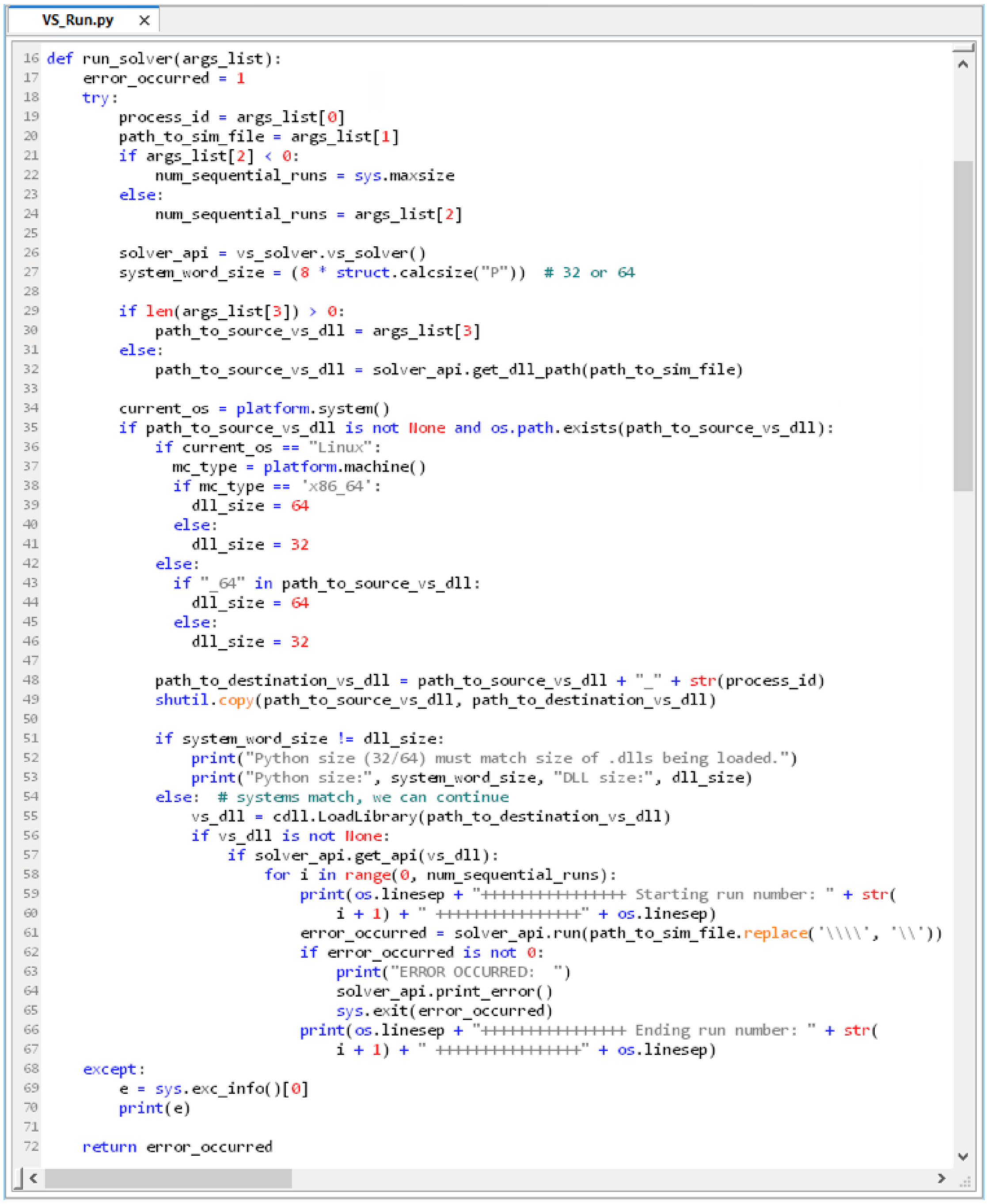
From the **Run Control** screen (Figure 1), click the button **Generate Files for this Run** (1) to generate the Simfile in the specified directory. After that, you can run the VB program from MSVS, or by double clicking on the EXE file as viewed in the Windows File Explorer.

Running a VS Math Model from Python

A wrapper program can also be written in Python to load the VS Solver and make a run.

The Python running the wrapper should match the DLLs being executed. That is, if you are referencing 32-bit DLLs, then you should run a 32-bit Python.

The example code in `VS_Run.py` (Figure 9) runs a simulation using the same general method shown for MATLAB and ANSI C.


VS_Run.py

```python
def run_solver(args_list):
    error_occurred = 1
    try:
        process_id = args_list[0]
        path_to_sim_file = args_list[1]
        if args_list[2] < 0:
            num_sequential_runs = sys.maxsize
        else:
            num_sequential_runs = args_list[2]

        solver_api = vs_solver.vs_solver()
        system_word_size = (8 * struct.calcsize("P"))  # 32 or 64

        if len(args_list[3]) > 0:
            path_to_source_vs_dll = args_list[3]
        else:
            path_to_source_vs_dll = solver_api.get_dll_path(path_to_sim_file)

        current_os = platform.system()
        if path_to_source_vs_dll is not None and os.path.exists(path_to_source_vs_dll):
            if current_os == "Linux":
              mc_type = platform.machine()
              if mc_type == 'x86_64':
                dll_size = 64
              else:
                dll_size = 32
            else:
              if "_64" in path_to_source_vs_dll:
                dll_size = 64
              else:
                dll_size = 32

            path_to_destination_vs_dll = path_to_source_vs_dll + "_" + str(process_id)
            shutil.copy(path_to_source_vs_dll, path_to_destination_vs_dll)

            if system_word_size != dll_size:
                print("Python size (32/64) must match size of .dlls being loaded.")
                print("Python size:", system_word_size, "DLL size:", dll_size)
            else:  # systems match, we can continue
                vs_dll = cdll.LoadLibrary(path_to_destination_vs_dll)
                if vs_dll is not None:
                    if solver_api.get_api(vs_dll):
                        for i in range(0, num_sequential_runs):
                            print(os.linesep + "++++++++++++++++ Starting run number: " + str(
                                i + 1) + " ++++++++++++++++" + os.linesep)
                            error_occurred = solver_api.run(path_to_sim_file.replace('\\\\', '\\'))
                            if error_occurred is not 0:
                                print("ERROR OCCURRED:  ")
                                solver_api.print_error()
                                sys.exit(error_occurred)
                            print(os.linesep + "++++++++++++++++ Ending run number: " + str(
                                i + 1) + " ++++++++++++++++" + os.linesep)
    except:
        e = sys.exc_info()[0]
        print(e)

    return error_occurred
```

Figure 9. Python program that loads VS Solver DLL and makes a run.

1. The name of a Simfile (the default is ”`simfile.sim`”) and number of runs to perform (default is 1) have been passed in as arguments are assigned to variables `path_to_sim_file` and `num_sequential_runs` respectively (lines 19 - 24).
2. An instance of the `vs_solver` class is assigned to the variable `solver_api` (line 26), and a pathname to the DLL file is obtained and assigned to the string variable `path_to_vs_dll` by scanning the specified Simfile (line 32).
3. If a file named in `path_to_vs_dll` exists (line 35), then the DLL is loaded (line 55) and some of the API functions are accessed (line 57).
4. The API function `vs_run` is applied using the `vs_solver` function `solver_api.run` (line 61).

To run from the VS Browser, link to a dataset in the **Models: Self-Contained Solvers** library (Figure 6), linked dataset shown in Figure 5, and check the box to use an external wrapper program ① and specify the pathname for the wrapper program ②. The `Steer_Control.py` example, included in the VS SDK, shows an example of this usage; the VS Browser runs Python via the Windows Command Script `Steer_Control_Py.cmd`. The version of Python used (32-bit/64-bit) should match that of the used VS Solver.

To run from the external environment, link to a dataset from the library **Models: Transfer to Local Windows Directory** ④ (Figure 1). In the linked dataset (Figure 2), specify the directory in which the Python file will be located ①. If you are not using a default name for the Simfile, specify a name, as was done earlier for the MATLAB example ③.

From the **Run Control** screen (Figure 1), click the button **Generate Files for this Run** ① to generate the Simfile in the specified directory. After that, you can run the Python program from your Python IDE, or by double clicking on the .py file as viewed in the Windows File Explorer.

Note The `vs_solver` class (as accessed with `vs_solver.py`) has been updated, as of Version 2020.1 from the old `Simulation` class. Users developing new code are encouraged to use the new `vs_solver` class, as the old class has been deprecated.