VS COM Interface

API of the VS Browser

Mechanical Simulation 

Table of Contents

Last revision: May 2022

Introduction

Mechanical Simulation Corporation produces and distributes software tools for simulating and analyzing the dynamic behavior of motor vehicles in response to steering, braking, and acceleration inputs. The simulation packages are organized into families of products named BikeSim®, CarSim®, SuspensionSim®, and TruckSim®. All are based on a technology called VehicleSim®.

VehicleSim Windows products all include a main program that provides a GUI for running the simulation models, visualizing results through animation and plotting, accessing documentation, and managing a database of vehicle and test descriptions. This main program (e.g., `CarSim.exe` in CarSim) is called the *VS Browser*. As with any Windows GUI, the VS Browser is normally controlled interactively using a mouse and keyboard. Alternatively, the VS Browser can work as a server that communicates with other programs using the Microsoft component object model (COM) interface. The VS Browser and COM interface are not available on Linux.

COM allows the VehicleSim product to be controlled with external software to make datasets and simulation runs, including those with hardware-in-the-loop. COM serves as the application program interface (API) for the VS Browser. You can program automation processes with other software tools that support COM, such as Visual Basic, MATLAB, Python, and so on.

This document describes API functions of the VS Browser that are accessed with COM. We assume that you know how to use COM from another software tool (e.g., MATLAB), and that you are familiar with the VS Browser structure and operation, as described in the VS Browser Reference Manual.

This manual provides a reference for the API of the VS Browser, and includes some examples showing how to connect with MATLAB and Visual Basic.

Register the Automation Server

Registration of a VS Browser as a COM server must be done manually. (This avoids ambiguities that might otherwise occur for users with multiple installations a VS Browser.)

After installing a VehicleSim product such as CarSim, a system administrator must run the Windows **Command Prompt** program (`cmd.exe`) to register the VS Browser as a COM automation server.

For example, to register CarSim, do the following:

1. Launch the Windows **Command Prompt** program as an Administrator.
2. Enter the command:

   {*CarSim prog folder*}\CarSim.exe **/RegServer**

If the pathname for the VS Browser contains spaces, then be sure to enclose it with double quotes, e.g.: `"C:\Program Files (x86)\CarSim901_Prog\CarSim.exe"`

This registration is needed only once. After the VS Browser is registered, you can start it at any time from COM client software.

Windows can only recognize one VehicleSim COM server program at a time. For example, if you have CarSim versions 8.2, 8.2.1, 8.2.2, 9.0, and 9.0.1 installed, only one of versions can be used as the COM server for CarSim at any given time. To change to a different version, you will need to repeat the registration using the pathname for the version you want to register.

There could be an unintended effect if you register a VS Browser using different users or register at multiple security levels (i.e., user level and admin level). We recommend you unregserver and then re-register the intended VS Browser installation again.

{*CarSim prog folder*}\CarSim.exe **/UnRegserver**

Starting a VS Browser as an Automation Server

In normal operation of a VehicleSim product, you launch the VS Browser by the same means as other Windows applications, such as using the Windows **Start** menu, or double-clicking an icon on your desktop. You can have multiple instances of a VS Solver running to view information from different databases at the same time. You can also run multiple versions of your VehicleSim product to visually compare information from different databases covering different versions.

When a VS Browser starts, it must initialize based on the selected database and possibly license settings used from a network license server. Based on your settings from the previous session with the VS Browser, you might be prompted when the VS Browser starts to choose a database and to choose or confirm license settings.

As noted in the previous section, if you have multiple versions of a VS Browser installed, only one will work at any given time as a COM server; this is the one most recently registered using the **/RegServer** option. Please also be aware that at any time, only one instance of the VS Browser can be a server.

Launch a VS Browser with a Command Line

You can manually launch the VS Browser with automation enabled with a command line using an optional parameter `Automation`. For example, with CarSim, use the command:

{*CarSim prog folder*}\CarSim.exe **/Automation**

Be aware that this will work only for the VS Browser that was registered most recently using the **RegServer** option.

Launch a VS Browser from External Software

Programs that support COM can launch a VS Browser that is registered as a COM server using the name associated with the product:

BikeSim:	"**BikeSim.Application**"
CarSim:	"**CarSim.Application**"

SuspensionSim: "**SuspensionSim.Application**"

TruckSim: "**TruckSim.Application**"

Here are some examples for starting CarSim from other languages.

From MATLAB:

```
h = actxserver('CarSim.Application')
```

From Visual Basic:

```
Dim h As Object
h = CreateObject("CarSim.Application")
```

From Python for Win32 (PythonWin):

```
import win32com.client
h = win32com.client.Dispatch("CarSim.Application")
```

In the above examples, `h` is a handle returned for use by the COM client.

If the activation command is repeated (e.g., `actxserver` in MATLAB), it will find that the requested server is already running and will share the control.

Be aware that if you launch a VS Browser from another program (e.g., MATLAB), then the VS Browser is launched as a thread under that program. If the controlling program is closed or uses the command to release the server, then the VS Browser will quit.

Special Considerations when Activating a VS Browser with COM

Once the VS Browser is running and showing the current dataset in the most recently used database, control by COM is straightforward. However, be aware that when the VS Browser initializes, it might popup windows and will not proceed until the user has responded.

Additionally, it is recommended that you do not rely on launching the VS Browser via the COM client dispatch command in a production environment. It is better practice to launch the application yourself, allowing you to apply command line parameters. You should also take care to ensure that there is only a single process associated with the registered executable. Multiple instances of the application running when the COM client attempts to establish the connection can result in unknown behavior.

Specify a Database and/or License Options

The VS Browser can be set up to show selection windows for specifying a database and/or license options. If set up this way, it will still do so when launched using a COM activation command (e.g., `actxserver`). In this case, the COM activation command will not complete until you have responded to the interactive prompts. These windows might be hidden behind your other software. For example, when starting from MATLAB, your MATLAB window might be in front of a prompt from the VS Browser that was launched from MATLAB.

If you wish to provide full automation from your external software (e.g., MATLAB), then run the VS Browser from the command line and include the optional parameter `SETQUICKLAUNCH`. For example, with CarSim, use the command:

```
{CarSim prog folder}\CarSim.exe /SETQUICKLAUNCH
```

This will disable the License and Database Selection dialogs from appearing when launching thereafter and will accept the default options. These settings can also be changed manually in the VS Browser.

License Expiration Notes

Be aware that if a license is within 45 days of expiring, a dialog box will appear the first time each day that a VS Browser is launched. If this notice appears when the VS Browser is launched using a COM activation command, the next COM command is likely to crash the Browser if it is sent before the dialog box has been addressed.

(Mechanical Simulation plans to fix this behavior for a future release.)

If it is not possible to renew the license, then a workaround is to launch the VS Browser once each day before starting the automation.

> **Note** If the license does expire, then the License Settings window comes up with no available license features.

Using a Fresh Database

When a VS database is opened for the first time with after an installation, the VS Browser automatically locks all Parsfiles by setting the Windows file permission to Read Only. If the VS Browser was launched using a COM activation command, the next COM command is likely to crash the Browser.

To avoid this, be sure to launch the VS Browser manually at least once to select the new database.

Using VS COM

VS COM supports most of the actions that you would take using GUI controls of the VS Browser. You can go to any dataset in any library. You can get and set values for yellow data fields, blue data links, drop-down controls, tables, etc, and can specify actions associated with the main buttons (**Plot**, **Video**, **Run Math Model**, etc.).

If you use external software to start the VS Browser as a COM server, be aware that it might prompt you to select a database and/or license options. For full automation, you can disable the options to make these selections when the VS Browser starts.

A typical VS client program will launch the VS Browser (e.g., CarSim), locate a **Run Control** dataset of interest, make a copy of the dataset, and possibly modify some properties of that dataset. The client program might repeat this with other datasets used by the run, such as

components of the vehicle model. When the desired changes have been made, the simulation is run.

The example database installed with your product has an example **Run Control** dataset **Baseline COM** in the category **External Control of Runs**. This is a copy of the double-lane change simulation used extensively in the Quick Start Guides for BikeSim, CarSim, and TruckSim. In the Quick Start Guide, the first variant on the baseline condition is a run with a different speed. The VehicleSim products also come with example external programs that use COM to automate the task of making the first run in the Quick Start Guide. They are located in the database folder, in `Extensions\RemoteControl`.

Details for the VS Browser API function are presented later. The descriptions that follow for the examples are intended only to show how COM is used to automate a basic procedure.

Identifying Libraries and Datasets

In the examples shown below, you will see that libraries and datasets are identified by the names that appear in the VS Browser. These names also appear in the Parsfiles associated with each dataset. One way to ensure that the names are spelled correctly is to view them in the Parsfile and use copy-paste to move them to the code you are writing in the external environment. For example, Figure 1 shows the Parsfile for a run that is created for the example described shortly. In line 2, the file has the Library name (1), the dataset name (2), and the category name (3). The three names are separated in the Parsfile by a back-quote character (`) that is used as a delimiter.

Identifying GUI Controls

Controls on the screen are identified by the unique keywords used to identify their values in the Parsfile. For example, lines 18-21 in Figure 1 show keywords and values for four checkboxes in the example dataset (4). The easiest way to see the keyword associated with a control on the screen is to view the screen and right-click on the control (the dataset must be unlocked to enable the right-click information). For example, Figure 2 shows the pop-up information obtained when you right-click on a checkbox on the **Run Control** screen (1); it indicates that the associated keyword is `#CheckBox3`.

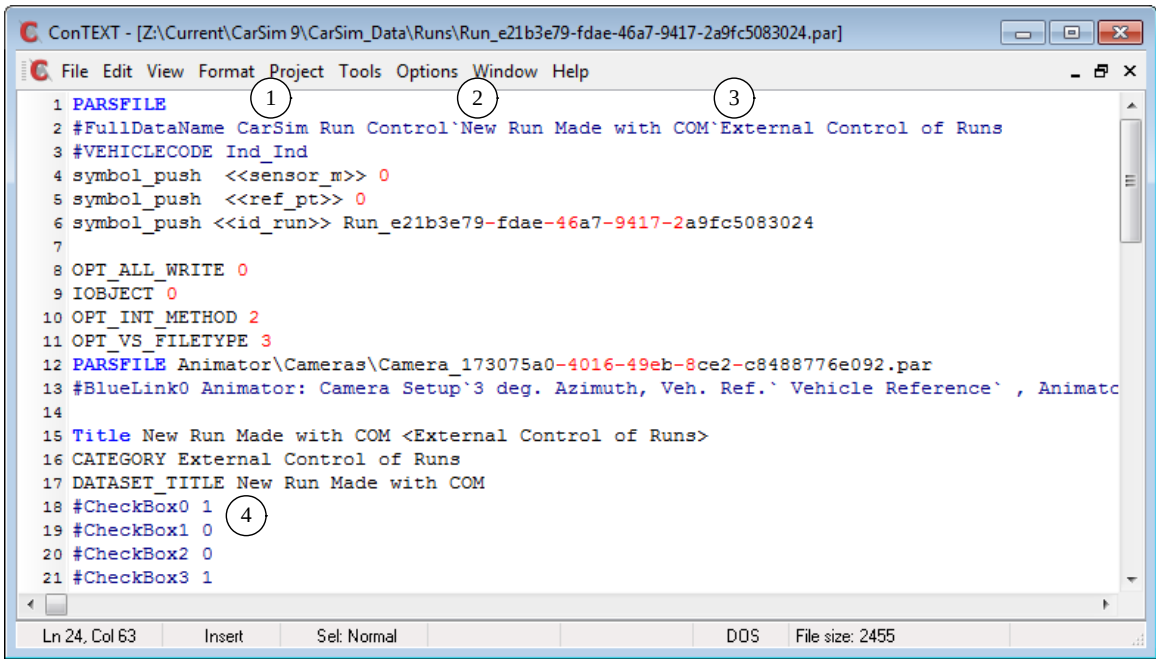


Figure 1. Use the Parsfile for an existing dataset to copy exact names.

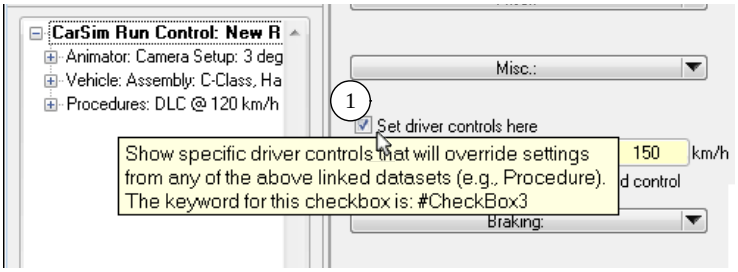


Figure 2. Right-click on a control to see its keyword.

MATLAB Example

Figure 3 shows the contents of a MATLAB M file that automates the creation and running of a new simulation. It works as follows:

1. Launch the VS Browser (CarSim in the example) using the MATLAB function `actserver`, and assign the result to a handle named `h` (line 2). All references to API functions in the VS Browser must include this handle. In the example, the convention followed is to use the format `h.`*function*.

2. The function `GoHome()` is used to perform the equivalent of clicking the Home button (line 3). This ensures that the **Run Control** library is active (in view).

3. If you want to see the definitions of the functions available for the server, then use the MATLAB `invoke` function (line 5, commented out).

4. Check for the existence of a dataset named `'Baseline COM'` in the category `'External Control of Runs'` (line 8). If it does not exist, stop now (line 10). Notice that in this example, this is the only time where an API function (`h.DataSetExists`) returns a value. That value is assigned to a variable `ok`, whose value indicates whether the dataset exists.

5. Delete the dataset `'New Run Made with COM'` in the category `'External Control of Runs'` if it exists (line 14). If this dataset does not exist, then the function has no effect other than possibly bringing a different dataset into view.

6. Go to the baseline dataset, duplicate it, and give it a new name (lines 17 – 19).

7. Make sure two checkboxes are checked (lines 22 and 23), select the speed control mode for a constant target speed (line 24), and specify a new speed (line 25). The API functions `Checkbox`, `Ring`, and `Yellow` identify specific controls on the screen using keywords associated with the controls (`#Checkbox8`, `#Checkbox3`, etc.), which were found by right-clicking on the controls of interest (Figure 2).

8. Run the simulation with the `Run` function (line 28).

9. Uncheck the checkbox for overlaying plots and animations (line 29) and launch the plotter tool (line 30).

10. If you want the VS Browser to quit, use the MATLAB `release` function (line 32, commented out).

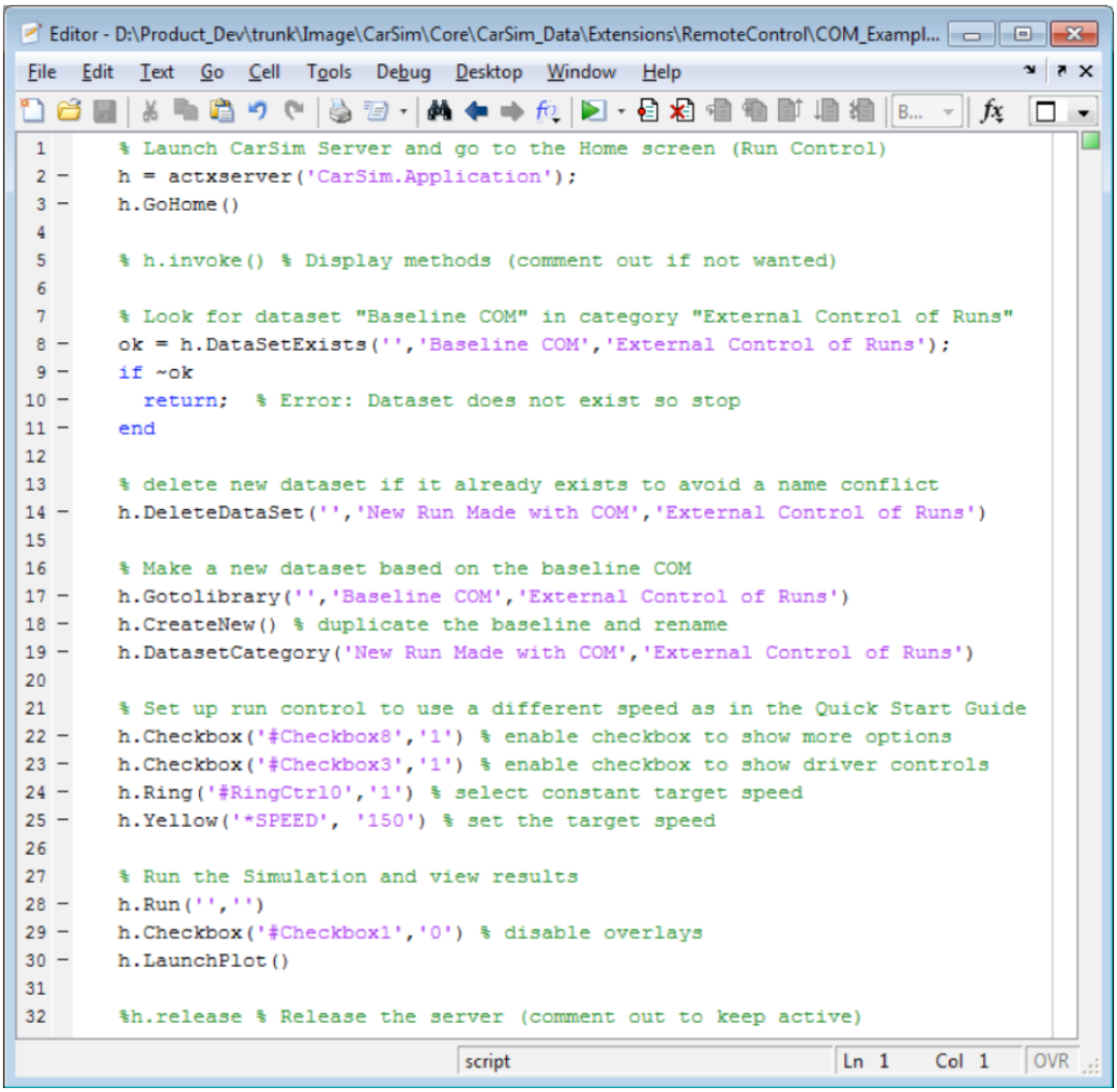


Figure 3. Control of VS Solver from MATLAB.

Visual Basic Example

Figure 4 shows a Visual Basic (VB) program that performs the same automation as was just shown for MATLAB.

1. Launch the VS Browser (CarSim in the example) using the VB function `CreateObject` and assign the result to a handle named `h` (line 3). All references to API functions in the VS Browser use the format `h.`*function*.

2. The function `GoHome()` is used to perform the equivalent of clicking the Home button (line 5). This ensures that the **Run Control** library is active.

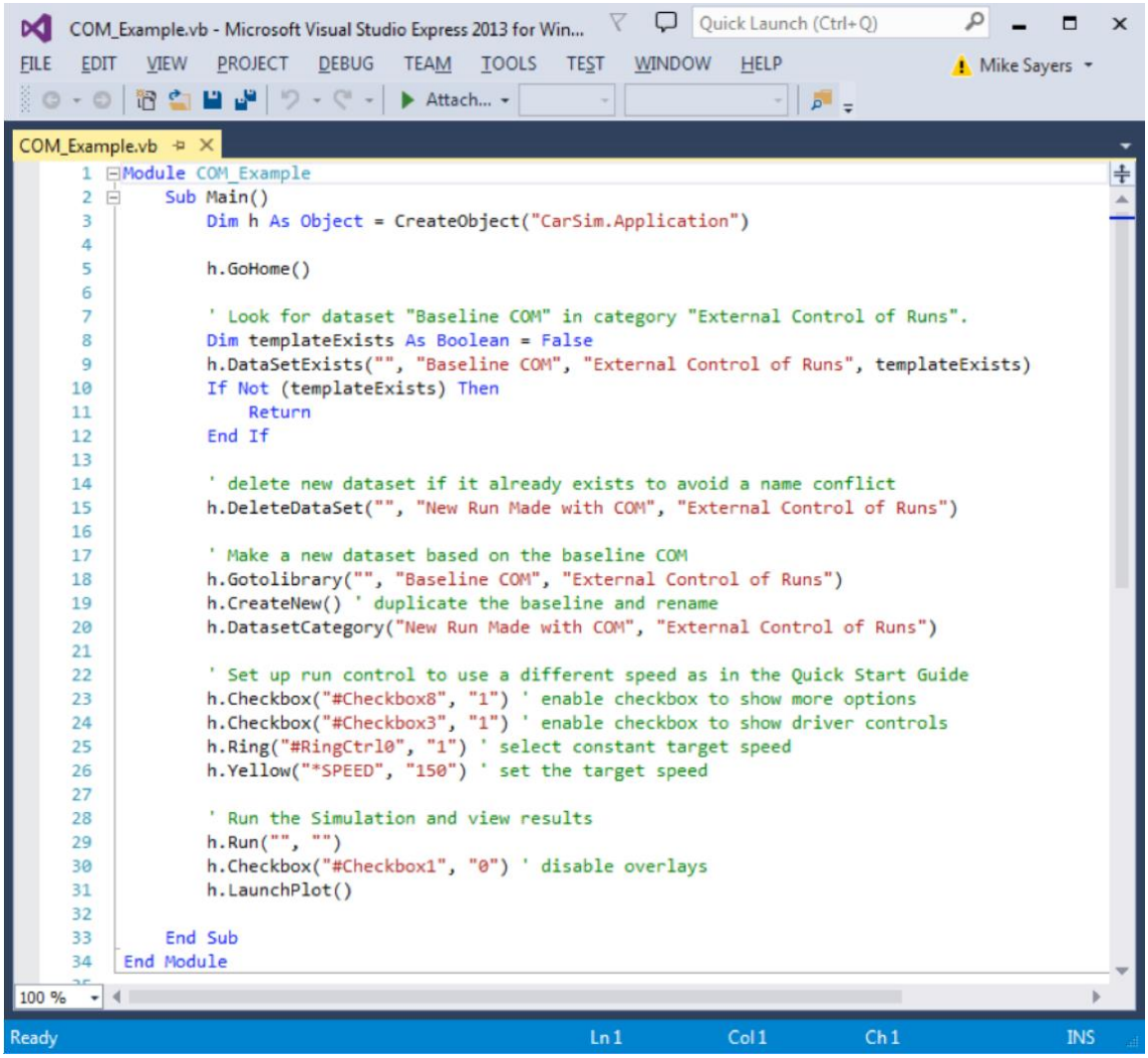


Figure 4. Control of VS Solver from Visual Basic (VB).

3. Check for the existence of a dataset named `'Baseline COM'` in the category `'External Control of Runs'` (line 9). If it does not exist, stop now (line 10). Notice that in this example, the function `h.DataSetExists` updates the value of the last argument, a Boolean named `templateExists`.

4. Delete the dataset `'New Run Made with COM'` in the category `'External Control of Runs'` if it exists (line 15). If this dataset does not exist, then the function has no effect other than possibly bringing a different dataset into view.

5. Go to the baseline dataset, duplicate it, and give it a new name (lines 18 – 20).

6. Make sure two checkboxes are checked (lines 23 and 24), select the speed control mode for a constant target speed (line 25), and specify a new speed (line 26). The API functions `Checkbox`, `Ring`, and `Yellow` identify specific controls on the screen using keywords associated with the controls (`#Checkbox8`, `#Checkbox3`, etc.), which were found by right-clicking on the controls of interest (Figure 2).

7. Run the simulation with the `Run` function (line 29).
8. Uncheck the checkbox for overlay plots and animations (line 30) and launch the plotter tool (line 31).

Because the VB module `Sub Main` launches the VS Browser (line 3), the Browser is automatically killed when the `Sub` returns.

COM Interface Functions

This section describes all of the COM functions, covering three categories: getting values, setting values, and miscellaneous actions (navigation, launching programs, creating datasets, etc.)

As shown in the examples, datasets and libraries are identified by names. Library names are fixed in the VS Browser design (e.g., `"Procedures"`). To reference a library, use the name exactly as it appears in the **Libraries** menu.

Note	Some libraries have long names that are shortened when shown in the Windows title area. For example, the CarSim library name **Suspensions: Independent Compliance, Springs, and Dampers** is shortened to **Susp: Independent Compliance** for display in the window title. (This is done to avoid cropping dataset titles.) In these cases, do not use the shortened name; it won't work.

Many of the functions interact with a user control (yellow field, data link, etc.). These are always accessed by the keyword that is written to the Parsfile to save the value of the control. To find the keyword for a specific control, right-click on the control to see the pop-up help (see Figure 2, page 8).

If a VS COM function is given invalid names or keywords, the function typically does nothing.

Syntax of VS COM Interface Functions

As shown by the examples in the previous section, the VS COM functions are accessed by activating a VS Browser as a server and assigning it to a server handle. The syntax for calling a function is *handle.fname* , where *handle* is the name of the handle of the server (e.g., this is a variable named '`h`' in the examples shown in Figure 3 and Figure 4).

Some VS COM functions set values for names or values; others obtain names and values; while others perform miscellaneous actions. Different languages handle the return of multiple values with their own syntax.

For example, languages such as VS and C/C++ support pointers to variables, and use syntax where the argument list of the function includes both inputs and outputs. In VB, the three names returned by `GetCurrentLibInfo` would be obtained with a statement involving variables that had been declared with type `Object`, e.g.,

```
Dim lib_name, dataset, cat As Object
h.GetCurrentLibInfo(lib_name, dataset, cat)
```

In the following descriptions of the VS COM functions, the form for function *fname* that has both input and output arguments is shown as:

fname(*type_i1 i1*, …, `[out]` *type_o1 o1*, …)

For each input, the form gives the type and name of the input argument. For each output, the form shows the identifier `[out]` followed by the type and name of the output argument. VS COM functions follow the convention that all input arguments appear before any of the output arguments.

For most languages, the argument list in the program will match the form shown in the following subsections. The type specified for outputs must be set using the options supported by the language. For example, in VB outputs are typically declared with the `Object` type as shown above.

On the other hand, in MATLAB there are three forms for applying a function, depending on how many outputs are returned:

1. If the function has no output arguments, the function is written on the line without an assignment, e.g., line 3, Figure 3:

```
h.GoHome()
```

2. If the function has a single output, it is written to assign the function to a single variable, e.g., line 8, Figure 3:

```
ok = h.DataSetExists('','Baseline COM','External Control of Runs')
```

3. If the function has multiple outputs, it is written to assign an array of variables, e.g.:

```
[lib, title, cat] = h.GetCurrentLibInfo()
```

In the following descriptions, the types specified for the arguments to the functions are specific to COM.

- `BSTR` is a Microsoft “Basic String” type used for COM communication.
- `VARIANT` is a Microsoft structure that is intended to allow more complex structures to be sent between programs written in different languages. Most of the VS COM functions that return strings use `VARIANT` pointers in the definitions.
- `SAFEARRAY` is a Microsoft type used for 1D or 2D matrices.
- Numerical arguments are documented using definitions from ANSI C.

Functions to Get Names and Values

Table 1 lists the VS COM functions that are used to get names of datasets and files, text for yellow fields, and states of controls.

Table 1. VS COM interface functions to get names, text data, and control states.

Function	Description
DataSetExists	Determine if a dataset exists with specified names.
GetBlueLink	Get information from a blue link in the current dataset.
GetCheckBox	Get the state of a checkbox on the currently active dataset.
GetCurrentDataSetID	Get the Parsfile UUID string for the currently active dataset.
GetCurrentLibInfo	Get names (library, dataset, category) for the current dataset.
GetCurrentParsfileName	Get the full pathname of the Parsfile for the current dataset.
GetDatabaseFolder	Get the working folder of the currently active database.
GetDataSetFolder	
GetDataSetID	Get the Parsfile UUID string of a given dataset.
GetDataSetInfoByIndex GetDataSetInfoByIndexShort	Get info (ID, Path, Category, Title) for given dataset.
GetDatasetList	Get a list of all datasets in a library.
GetEmbeddedNotes	Get the notes field in the current screen.
GetMiscYellow	Get the text from a misc. yellow field in the current dataset.
GetNumDatasets GetNumDatasetsShort	Get the number of datasets in a given library.
GetParsfileName	Get the Parsfile name of a given dataset.
GetPathname	Get text from a pathname field in the current dataset.
GetProgFolder	Get the name of the Prog folder that contains the VS Browser.
GetRing	Get the current value of a ring control (drop-down list).
GetTable	Get the array of numbers for a table the current dataset.
GetYellow	Get the text from a yellow field in the current dataset.
GetYellow2	Get the text from a yellow field in a given dataset.
GetMixTableNumRows	Get the number of rows in a MixTable.
GetMixTableNumColumns	Get the number of internal columns in a MixTable.
GetMixTableCellValue	Get the value of a cell in a MixTable.
GetMixTableColumnTypeDescription	Get the content type for an internal column of a MixTable.
GetMixTableColumnHelpText	Get the help text associated with an internal column in a MixTable.
QueryRunForResults	Get a list of results available for the requested run.
GetError	Gets the detailed error information of a previous operation that has failed.

DataSetExists

Form:

```
DataSetExists(BSTR Library, BSTR DataSet, BSTR Category,
              [out] VARIANT_BOOL * Exists)
```

Inputs:

`Library:` Library name.

`DataSet:` Dataset name.

`Category:` Category name.

Output:

`Exists:` Boolean: 1 if the dataset exists; 0 if not.

GetBlueLink

Form:

```
GetBlueLink(BSTR Keyword, [out] VARIANT * Library,
      [out] VARIANT * DataSet, [out] VARIANT * Category,
      [out] VARIANT * FileName)
```

Input:

`Keyword:` Keyword for a data link. All keywords have the form: `#BlueLink`*ID*, where *ID* goes from 0 to *n-1* (*n* is the number of links on the screen). You can obtain this keyword interactively by right-clicking on the link of interest.

Outputs:

`Library`: Name of library containing the dataset for the link.

`DataSet`: Name of dataset for the link.

`Category`: Category name for the dataset for the link.

`FileName`: Parsfile name for the dataset for the link.

GetCheckbox

Form:

```
GetCheckbox(BSTR Keyword, [out] short * State)
```

Input:

`Keyword:` Keyword for the checkbox. All keywords have the form: `#CheckBox`*ID*, where *ID* goes from 0 to *n-1* (*n* is the number of checkboxes on the screen). In addition, some checkboxes might have alternate keywords that are also recognized, if indicated when right-clicking on the control.

Output:

`State:` Value for the checkbox: 0 or 1. If the specified checkbox cannot be found, the returned value is –1.

GetCurrentDataSetID

Form:

```
GetCurrentDataSetID([out] VARIANT * ID)
```

Output:

`ID:` File ID (library prefix_[UUID]).

Get the Parsfile name for the current dataset, up to the period. For example, if the dataset currently in view is associated with the Parsfile named `Run_e21b3e79-fdae-46a7-9417-2a9fc5083024.par`, this function would return the string: `"Run_e21b3e79-fdae-46a7-9417-2a9fc5083024"`.

GetCurrentLibInfo

Form:

```
GetCurrentLibInfo([out] VARIANT * Library,
                  [out] VARIANT * DataSet,
                  [out] VARIANT * Category)
```

Outputs:

`Library:` Library name.

`DataSet:` Dataset name.

`Category:` Category name.

GetCurrentParsfileName

Form:

```
GetCurrentParsfileName([out] VARIANT * pathname)
```

Output:

`pathname:` Full pathname for the Parsfile of the dataset currently in view.

GetDatabaseFolder

Form:

```
GetDatabaseFolder([out] VARIANT * folder)
```

Output:

`folder:` Folder name of the current working database.

GetDataSetFolder

This is identical in form and behavior to the `GetDatabaseFolder` function. This is a legacy name. It precedes the current VS terminology in which the root folder is called the database.

GetDataSetID

Form:

```
GetDataSetID(BSTR Library, BSTR DataSet, BSTR Category,
             [out] VARIANT * ID)
```

Inputs:

`Library:` Library name.

`DataSet:` Dataset name.

`Category:` Category name.

Output:

`ID:` File ID (library prefix_[UUID]), or empty string if the specified dataset does not exist.

Get the Parsfile ID string of a specified dataset that is not necessarily the currently active one.

GetDataSetInfoByIndex

GetDataSetInfoByIndexShort

Form:

```
GetDataSetInfoByIndex(BSTR Library, unsigned long Index,
             BOOL FullPath, [out] VARIANT * ID,
             [out] VARIANT *FilePath, [out] VARIANT *Category,
             [out] VARIANT *Title)
```

Inputs:

`Library:` Library name.

`Index:` Index of dataset to retrieve, 0 to n-1, where n is the number of datasets in the library.

`FullPath:` If TRUE, the global pathname will be returned in `FilePath`. Otherwise, the relative path within the database will be returned.

Outputs:

`ID:` File ID (library prefix_[UUID]).

`FilePath:` Either the global or relative path of the requested dataset (based on the value of `FullPath`), or blank if the specified dataset does not exist.

`Category:` Dataset category.

`Title:` Dataset title.

This function information about a dataset in a library based on the numerical `Index`, which is between 0 and n-1, where n is the number of datasets in the library. If the library name is not valid or `Index` is out of range, the outputs are not defined. (The values depend on the language being used.)

The "Short" version of this function outputs `Index` as a short value, as some older environments do not support unsigned types.

GetDatasetList

Form:

```
GetDatasetList(BSTR Library, [out] SAFEARRAY ** List)
```

Note The naming of this function is different than others: it includes "`Dataset`" (with a lowercase 's') as opposed to all other instances that use "`DataSet`" (with an uppercase 'S'). This was a typo made when the function was introduced in 2007. It has been kept this way to maintain backward compatibility with existing programs.

Input:

`Library:` Library name.

Output:

`List:` Array of strings, where each string gives label information for a dataset: file ID, followed by category name, followed by dataset name (returned). The format is:

id:< *category name* > *dataset name*

For example, Listing 1 shows example results obtained from MATLAB for a CarSim library.

Listing 1. List of datasets in a library.

```
>> h.GetDatasetList('Powertrain: Engine')

ans =

    'Engine_6d2a4e9b-2...1ea:<CS Engine Torque Curves>125 kW  Engine'
    'Engine_5fd8cccf-3...e86:<CS Engine Torque Curves>150 kW  Engine'
    'Engine_1b115f1d-8...974:<CS Engine Torque Curves>150 kW  Engine Off Motoring …
    'Engine_d49c24d0-f...7ce:<CS Engine Torque Curves>200 kW  Engine'
    'Engine_3c6aca1f-e...b90:<CS Engine Torque Curves>250 kW  Engine'
    'Engine_1b47c442-1...419:<CS Engine Torque Curves>270 kW Diesel Engine'
    'Engine_8040b1ad-5...fc3:<CS Engine Torque Curves>300 kW  Engine'
    'Engine_3c07daf9-8...fab:<CS Engine Torque Curves>300 kW  Engine (GT)'
    'Engine_cd75601b-3...782:<CS Engine Torque Curves>75 kW  Engine'
>>
```

GetEmbeddedNotes

Form:

`GetEmbeddedNotes ([out] VARIANT * outNotes)`

Output:

`outNotes:` Text from the current screen's notes field.

GetMiscYellow

Form:

`GetMiscYellow(BSTR Keyword, [out] VARIANT * Contents)`

Input:

`Keyword:` Keyword for a scrollable miscellaneous yellow field. All keywords have the form: `#MiscYellow`*ID*, where *ID* goes from 0 to *n-1* (*n* is the number of misc. fields on the screen). You can obtain this keyword interactively by right-clicking on the field of interest.

Output:

`Contents:` Text from the specified miscellaneous yellow field.

GetNumDatasets

GetNumDatasetsShort

Form:

`GetNumDatasets(BSTR Library, [out] unsigned long *count)`

Input:

`Library:` Library name.

Output:

`count:` Number of datasets within the library.

The "Short" version of this function outputs `count` as a short value, as some older environments do not support unsigned types.

GetParsfileName

Form:

```
GetParsfileName(BSTR Library, BSTR DataSet, BSTR Category, [out]
                VARIANT * Parsfile)
```

Inputs:

`Library:` Library name; if empty, the currently active library is used.

`DataSet:` Dataset name.

`Category:` Category name.

Output:

`Parsfile:` Parsfile name. If the library does not exist, this returns "Unknown Library"; if the dataset does not exist, this returns "Unknown Dataset".

GetPathname

Form:

```
GetPathname(BSTR Keyword, [out] VARIANT * Pathname)
```

Input:

`Keyword:` Keyword of a pathname field with an adjacent file browser button. You can obtain this keyword interactively by right-clicking on the pathname field of interest.

Output:

`Pathname:` Text specified in the pathname field of the current dataset.

GetProgFolder

Form:

```
GetProgFolder([out] VARIANT * Folder)
```

Output:

`Folder:` Folder that contains the VS Browser that is the current COM server (e.g., "`C:\Program Files (x86)\CarSim900_Prog`").

GetRing

Form:

```
GetRing(BSTR Keyword, [out] VARIANT * Value)
```

Input:

`Keyword:` Ring control (drop-down list) keyword. All keywords have the form: `#RingCtrl`*ID*, where *ID* goes from 0 to *n-1* (*n* is the number of ring controls on the screen). You can obtain this keyword interactively by right-clicking on the ring control of interest.

Output:

`Value:` String value of the specified ring control in the current dataset.

Note Two versions of ring controls are used in the VS Browser GUI. One is a stand-alone control with a keyword of the form `#RingCtrl`*ID*; the other is part of a Blue data link that is used to specify the library containing the dataset of the link, and has a keyword of the form `#BlueLink`*ID*.

If the control is part of a data link, then you must use the `GetBlueLink` function to obtain the value.

GetTable

Form:

```
GetTable(BSTR Keyword, [out] SAFEARRAY ** Data)
```

Input:

`Keyword:` Keyword for the table (also called a Configurable Function).

Output:

`Data:` 2D array of values for the table (returned).

Get the array of numbers in a specified table in the currently active dataset. This works for 1D and 2D tables.

Note If the table can be simplified to a constant or coefficient, you can get the value with the commands `GetRing` (to get the type of calculation:

constant, coefficient, 1D table, etc.) and the command `GetYellow` (to get the value from the yellow field that is used for the simplified representation).

GetYellow

Form:

```
GetYellow(BSTR Keyword, [out] VARIANT * Contents)
```

Input:

`Keyword:` Yellow field keyword. You can obtain this keyword interactively by right-clicking on the field of interest.

Output:

`Contents:` Text in the specified yellow field in the current dataset.

GetYellow2

Form:

```
GetYellow2(BSTR Library, BSTR DataSet, BSTR Category,
      BSTR Keyword, [out] VARIANT * Contents)
```

Inputs:

`Library:` Library name.

`DataSet:` Dataset name.

`Category:` Category name.

`Keyword:` Yellow field keyword.

Output:

`Contents:` Text in the specified yellow field.

Get the contents of a yellow field in the given library/dataset/category name. If the library or dataset cannot be found, then this returns a blank string.

GetMixTableNumRows

Form:

```
long GetMixTableNumRows(BSTR tableName)
```

Inputs:

`tableName:` The name of the MixTable (e.g. #MixTable0).

Returns the number of rows in the requested MixTable. Unlike the MixTable columns, the rows should always match what is displayed to the user via the VS Browser.

GetMixTableNumColumns

Form:

```
long GetMixTableNumColumns(BSTR tableName)
```

Inputs:

`tableName:` The name of the MixTable (e.g. #MixTable0).

Returns the number of columns in the requested MixTable. This is the total number of columns that could be displayed and may not match what is shown through the VS Browser. Certain columns can be hidden for a row based on the selected values of other cells in the given row.

GetMixTableCellValue

Form:

```
BSTR GetMixTableCellValue(BSTR tableName, long column, long row)
```

Inputs:

`tableName:` The name of the MixTable (e.g. #MixTable0).

`column:` The zero-based index into the internal column.

`row:` The zero-based index into the internal row.

Returns a string with the value of the requested cell.

Please note that the column and row are zero-based. `GetMixTableCellValue` returns the contents of the cell even if it is not displayed in the VS Browser. Some data cells are hidden based on column dependencies on other cell values. The screen parsfile lists the internal column definitions before the stored data for a given MixTable. You can also right-click on a particular cell to determine which column needs to be passed to this function to retrieve the proper value.

GetMixTableColumnTypeDescription

Form:

```
BSTR GetMixTableColumnTypeDescription(BSTR tableName, long
      column)
```

Inputs:

`tableName:` The name of the MixTable (e.g. #MixTable0).

`column:` The zero-based index into the internal column.

Returns a string with a description of the type of cell contained in the specified internal column.

GetMixTableColumnHelpText

Form:

```
BSTR GetMixTableColumnHelpText(BSTR tableName, long column)
```

Inputs:

`tableName:` The name of the MixTable (e.g. #MixTable0).

`column:` The zero-based index into the internal column.

Returns a string with the help text associated with the specified internal column.

MixTable Example: Segment Builder

The Path: Segment Builder MixTable has 14 columns that contain data for the five types of segments that can be used to define a path. Table 2 below illustrates which columns contain data for each segment type. For each row, only the cells relevant to the segment type are displayed in the VS Browser. For example, if a radius segment is added to the path on the Segment Builder screen, columns 4 – 7 are displayed, while columns 1 – 3 and columns 8 – 13 are hidden. Table 3 below lists the data required for each segment type and the column where it is stored.

Table 2. Segment Builder MixTable Columns

C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13
Table													
Straight													
Radius													
Curvature													
Clothoid													

Table 3. Segment Builder Input Data

Segment Type (Column 0)	Columns used for segment type	Description of data stored in column
Table	Column 1	Link to a dataset from the X-Y Coordinates for Segment library
Straight	Column 2	Length of straight segment in meters
	Column 3	Unit label – meters (m)
Radius	Column 4	Segment radius in meters
	Column 5	Unit label – meters (m)
	Column 6	Length of segment arc, specified with arc length or central angle
	Column 7	Specify degrees for central angle (deg) or meters for arc length (m)
Curvature	Column 8	Segment curvature (inverse of radius, 1/m)
	Column 9	Unit label for segment curvature (1/m)
	Column 10	Length of segment arc, specified

		with arc length or central angle
	Column 11	Specify degrees for central angle (deg) or meters for arc length (m)
Clothoid	Column 12	Length of clothoid segment in meters (m)
	Column 13	Unit label – meters (m)

QueryRunForResults

Form:

```
QueryRunForResults(BSTR runID, [out] SAFEARRAY **list)
```

Input:

`runID:` The name of the run to query (e.g. Run_27872fb3-832e-4e6b-a577-9faf134cb57a).

Output:

`list:` Array of strings, where each string is the name of a results file.

Returns a list of results available for the requested run.

GetError

Form:

```
GetError([out] long * Code, [out] BSTR * Message)
```

Outputs:

`Code:` The integer error code of the error.

`Message:` A string message associated with the error.

Gets the detailed error information of a previous operation that has failed, if any. COM functions ending in *_CheckError* always write this error information. Other COM functions may, but are not guaranteed to write error information. Error information is context-specific and depends on the underlying operation that failed. For example, a failed solver run will return a non-zero `Code` and a `Message` describing the specific problem. For successful execution, the error information will contain a `Code` of 0 and a blank string for `Message`.

Functions to Set Names and Values

Table 4 lists the VS COM functions that are used to set names of datasets and values of controls. These functions have only input arguments.

Table 4. VS COM interface functions to set names and values.

Function	Description
`BlueLink`	Set a (blue) data link in the current dataset.
`Checkbox`	Set a checkbox value in the current dataset.
`DatasetCategory`	Set dataset and category names for the current dataset.
`MiscYellow`	Replace a miscellaneous yellow field in the current dataset.
`MiscYellowInsert`	Insert a string in a misc. yellow field in the current dataset.
`AddEmbeddedNote`	Add a string to the notes field in the current screen.
`SetEmbeddedNotes`	Set the notes field in the current screen.
`Radio`	Set a radio control value in the current dataset.
`Ring`	Set a ring control (drop-down list) value in the current dataset.
`SetPathname`	Set path and file name.
`SetTable`	Set table value in the current dataset.
`Yellow`	Set the contents of a yellow field in the current dataset.
`SetMixTableCellValue`	Set the value of the given cell in a MixTable.
`SetMixTableNumRows`	Set the number of rows in a MixTable.

BlueLink

Form:

```
BlueLink(BSTR Keyword, BSTR Library, BSTR DataSet, BSTR Category)
```

Inputs:

`Keyword`: Keyword for a data link. All keywords have the form: `#BlueLink`*ID*, where *ID* goes from 0 to *n-1* (*n* is the number of links on the screen). You can obtain this keyword interactively by right-clicking on the link of interest.

`Library`: Name of library containing the dataset for the new link. If this string is empty, the link is cleared.

`DataSet`: Name of dataset for the new link. If this string is empty, the link is cleared.

`Category`: Category name for the dataset for the new link.

Modify a blue link on the current screen. If either `Keyword` or the specified dataset does not exist, then this command has no effect. If either the library title or the dataset title is blank, then the specified blue link is cleared (no link).

Checkbox

Form:

```
Checkbox(BSTR Keyword, BSTR Value)
```

Inputs:

`Keyword:` Keyword for the checkbox. All keywords have the form: `#CheckBox`*ID*, where *ID* goes from 0 to *n-1* (*n* is the number of checkboxes on the screen). In addition, some checkboxes might have alternate keywords that are also recognized, if indicated when right-clicking on the control.

`Value:` Value for the checkbox: "0" or "1".

Modify a checkbox value in the current dataset. If `Keyword` is not valid, then this command has no effect.

DatasetCategory

Form:

```
DatasetCategory(BSTR DataSet, BSTR Category)
```

Inputs:

`DataSet:` New dataset name assigned to the current dataset.

`Category:` New category name assigned to the current dataset.

MiscYellow

Form:

```
MiscYellow(BSTR Keyword, BSTR Contents)
```

Inputs:

`Keyword:` Keyword for a scrollable miscellaneous yellow field. All keywords have the form: `#MiscYellow`*ID*, where *ID* goes from 0 to *n-1* (*n* is the number of misc. fields on the screen). You can obtain this keyword interactively by right-clicking on the field of interest.

`Contents:` Contents (text) to put into the specified miscellaneous yellow field.

Replace the contents of a miscellaneous yellow field in the current dataset. If `Keyword` is not valid, then this command has no effect.

MiscYellowInsert

Form:

```
MiscYellowInsert(BSTR Keyword, short Index, BSTR Text)
```

Inputs:

`Keyword:` Keyword for the field. All keywords have the form: `#MiscYellow`*ID*, where *ID* goes from 0 to *n-1* (*n* is the number

of misc. fields on the screen). You can obtain this keyword interactively by right-clicking on the field of interest.

`Index:` Zero-based index into the lines of text in the field that determines where the text will be inserted. Use –1 to insert the text at the end of the existing contents of the field.

`Text:` String of text to be inserted into the field, after any existing text.

Insert text into a miscellaneous yellow field in the current dataset. The new text is appended to any text that already exists in the field. If `Keyword` is not valid, then this command has no effect.

AddEmbeddedNote

Form:

```
AddEmbeddedNote(BSTR Text)
```

Inputs:

`Text:` String of text to be inserted into the field, after any existing text.

Insert text into the notes field on the left sidebar of the current screen. This will also save the text into the `#EMBEDDED_NOTES` segment of the parsfile.

SetEmbeddedNotes

Form:

```
SetEmbeddedNotes(BSTR Text)
```

Inputs:

`Text:` String of text to be inserted into the field, replacing any existing text.

Set text in the notes field on the left sidebar of the current screen. This will also save the text into the `#EMBEDDED_NOTES` segment of the parsfile.

Radio

Form:

```
Radio(BSTR Keyword, BSTR Value)
```

Inputs:

`Keyword:` Keyword for a radio control. All keywords have the form: `#RadioCtrl`*ID*, where *ID* goes from 0 to *n-1* (*n* is the number of radio controls on the screen).

`Value:` Value to set for the control.

Ring

Form:

```
Ring(BSTR RingID, BSTR Value)
```

Inputs:

`Keyword:` Ring control (drop-down list) keyword. All keywords have the form: `#RingCtrl`*ID*, where *ID* goes from 0 to *n-1* (*n* is the number of ring controls on the screen). In addition, some ring controls might have alternate keywords that are also recognized, if indicated when right-clicking on the control.

`Value:` Value to set for the ring control.

Set the value of a ring control (drop-down list) in the current dataset. If `Keyword` is not valid, then this command has no effect.

> **Note** Two versions of ring controls are used in the VS Browser GUI. One is a stand-alone control with a keyword of the form `#RingCtrl`*ID*; the other is part of a Blue data link that is used to specify the library containing the dataset of the link, and has a keyword of the form `#BlueLink`*ID*.
>
> If the control is part of a data link, then you must use the `BlueLink` function to set the value.

SetPathname

Form:

```
SetPathname(BSTR Keyword, BSTR Pathname)
```

Input:

`Keyword:` Keyword of a pathname field with an adjacent file browser button. You can obtain this keyword interactively by right-clicking on the pathname field of interest.

`Pathname:` Full pathname you are going to set.

Set the contents of a pathname field in the current dataset. If `Keyword` is not valid, then this command has no effect.

SetTable

Form:

```
SetTable(BSTR Keyword, SAFEARRAY * Data)
```

Input:

`Keyword:` Keyword for the table (also called a Configurable Function).

`Data:` Two-dimensional array of double precision numbers. The numbers should be arranged by rows: row 1, row 2, etc. This array will replace the array of the specified table control.

Set the contents of a table in the current dataset. If `Keyword` is not valid, then this command has no effect.

Yellow

Form:

```
Yellow(BSTR Keyword, BSTR Value)
```

Inputs:

`Keyword:` Yellow field keyword. You can obtain this keyword interactively by right-clicking on the field of interest.

`Contents:` Contents to place into the yellow field in the current dataset.

Set the contents of a yellow field in the current dataset. If `Keyword` is not valid, then this command has no effect.

SetMixTableCellValue

Form:

```
SetMixTableCellValue(BSTR tableName, long column, long row, BSTR
cellValue)
```

Inputs:

`tableName:` The name of the MixTable (e.g. #MixTable0).

`column:` The zero-based index into the internal column.

`row:` The zero-based index into the internal row.

`cellValue:` The new value of the cell.

Set the contents of a cell within a MixTable. Please note that the column and row are zero-based, and get the contents of the cell even if it is not displayed in the GUI based on dependencies.

SetMixTableNumRows

Form:

```
SetMixTableNumRows(BSTR tableName,long numRows)
```

Inputs:

`tableName:` The name of the MixTable.

`numRows:` The new number of rows in the MixTable.

Sets the number of rows in a MixTable. New rows can be filled with default values, depending on the screen.

Functions to Perform Miscellaneous Actions

Table 5 lists the VS COM functions that are used to perform miscellaneous actions (navigation, launching programs, creating datasets, etc.)

Table 5. VS COM interface functions for miscellaneous actions.

Function	**Description**
`BatchRun`	Make a batch of runs using the **Batch Runs** library.
`BatchRun_CheckError`	Make a batch of runs (with output error flag).
`CreateAllParFile`	Create the run_all.par file for a specified example.
`CreateNew`	Create a new dataset.
`DeleteDataSet`	Delete a dataset.
`ExportParsfile`	Export expanded Parsfile from the current dataset.
`Goback`	Go back to the library viewed previously.
`GoNext`	Go to the next dataset in the current library.
`GoPrevious`	Go to the previous dataset in the current library.
`Gotolibrary`	Go to a specified dataset in a specified library.
`ImportParsfile`	Import a previously exported Parsfile (any type).
`LaunchPlot`	Launch plot program; same as clicking the Plot button.
`Lock`	Lock the current dataset.
`LockAll`	Lock all of the datasets.
`Minimize`	Minimize the VS Browser window.
`Run`	Run a given dataset name and category name.
`Run_CheckError`	Run a given dataset and category (with output error flag).
`RunButtonClick`	Simulate a mouse click on the run button.
`Run_dSPACE`	Run a given dataset on the dSPACE platform.
`Run_SCALEXIO`	Run a given dataset on the dSPACE SCALEXIO platform.
`CreateAllParFile`	Create the run_all.par file for the given dataset and category.
`StartAnimator`	Launch playback animator.
`StartAnimatorAndPlot`	Launch playback animator with plot.
`StartLiveVideo`	Launch live animator.
`CreateAnimatorParFile`	Create the animator.par file for the current dataset.

Unlock	Unlock the current dataset.
UnlockAll	Unlock all of the datasets.
Unminimize	Restore minimized the GUI window.
ReadGeoLocations	Imports GPS coordinate file on the **Path: X-Y** screen.
UpdateRoad3DShape	Update the Road Shape Files on the **Road: 3D Surface** screen.
SetOverlayPlotFile	Set the CSV, ERD, or VS file for plot overlay.
RewriteReindex	Reindex and Rewrite all pars files in the database.
ReindexDatabase	Reindex the database.
RewriteDatabase	Rewrite all pars files in the database.
RewriteAllRuns	Rewrite all run_all.par files.
Run_Background	Run a simulation in background state.
Run_dSPACE_Background	Run a given dataset on the dSPACE platform in background.
Set_RundSPACE	Pause, play, and run after executing Run_dSPACE.
StopWindowsRun	Stop the run manually.

BatchRun

Form:

```
BatchRun(BSTR DataSet, BSTR Category)
```

Inputs:

`DataSet:` Name of dataset from the **Batch Runs** library.

`Category:` Category name of dataset from the **Batch Runs** library.

This is the same as going to a specified dataset in the **Batch Runs** library and clicking the button **Make a Batch of Runs**. If the strings `DataSet` and `Category` are empty, the currently active dataset in the **Batch Runs** library will be used. If the dataset cannot be found, or if the specified dataset is locked, then no runs are made. Regardless of whether any runs are made, this command will still switch to the **Batch Runs** library.

BatchRun_CheckError

Form:

```
BatchRun_CheckError(BSTR DataSet, BSTR Category, [out]
VARIANT_BOOL * ThrewError)
```

Inputs:

`DataSet:` Name of dataset from the **Batch Runs** library.

`Category:` Category name of dataset from the **Batch Runs** library.

Output:

`ThrewError:` Flag indicating whether the batch run terminated with an error.

Run a batch simulation for the given **Batch Runs** dataset name and category name. This function is equivalent to the *BatchRun* function above but with an output flag, `ThrewError`, indicating whether the batch run terminated with an error. If `ThrewError` is true, call the *GetError*

function to retrieve detailed error information. Solver errors will terminate the entire batch run, with an error message describing the problematic run.

CreateAllParFile

Form:

```
CreateAllParFile (BSTR DataSet, BSTR Category)
```

Inputs:

`DataSet:` Name of dataset from the **Batch Runs** library.

`Category:` Category name of dataset from the **Batch Runs** library.

Generate the run_all.par file for a specified run. This will perform the same operation as clicking the Run button on the Run control screen. Clicking Run, will process all the linked parsfiles and generate the run_all.par.

CreateNew

Form:

```
CreateNew(VOID)
```

Create a new dataset by copying the current one and changing the name to ensure the new dataset has a unique title. Applying this function is the same as using the **File** menu item **New Dataset** (Ctrl+N) or clicking the **Duplicate** button in the toolbar.

DeleteDataSet

Form:

```
DeleteDataSet(BSTR Library, BSTR DataSet, BSTR Category)
```

Inputs:

`Library:` Name of library with dataset to be deleted. If this is blank, use the currently active library.

`DataSet:` Name of dataset to be deleted.

`Category:` Name of category of dataset to be deleted

Use this function to delete the specified dataset.

If the string `Library` is empty, the function will use the dataset within the current library. If the strings `DataSet` and `Category` are both empty, the current dataset will be deleted (in this case, it is the same as using the **File** menu item **Delete Dataset** (Ctrl+D) or clicking the **Delete** button in the toolbar).

ExportParsfile

Form:

```
ExportParsfile(BSTR FileName, short mode)
```

Inputs:

`Filename:`	File name of Parsfile with full path to export.
`mode:`	Mode of Parsfile format; 0 – plain text expanded Parsfile (.PAR), 1 – consolidated Parsfile (.CPAR), 2 – CPAR with preferences included, 3 – CPAR with run results included, 4 – CPAR with preferences and run results, 5 – CPAR with all runs.

Create a new file (*.PAR or *.CPAR) that contains all of the data from the current screen plus all of the linked data screens. If the extension is .CPAR the new file will also contain support files (images, animator shape files, etc.) contained in the database. The new file can be imported into another database to bring in everything needed to support the dataset currently displayed. Applying this function is the same as using the **File** menu item **Export Expanded Parsfile...** for exporting expanded Parsfile or **Export Consolidated Parsfile...** for exporting consolidated Parsfile.

Goback

Form:

```
Goback(VOID)
```

Go back to the library viewed previously. Applying this function is the same as using the **Libraries** menu item **Back** (Ctrl+Left) or clicking the **Back** button in the toolbar.

GoNext

Form:

```
GoNext(VOID)
```

Go to the next dataset in the current library. If the current dataset is the last one, this button takes you to the first one (the ordering is circular). Applying this function is the same as using the **Datasets** menu item **Next Dataset** (Ctrl+Up) or clicking the **Next** button in the toolbar.

GoPrevious

Form:

```
GoPrevious(VOID)
```

Go to the preceding dataset in the current library. If the current dataset is the first one, this button takes you to the last one (the ordering is circular). Applying this function is the same as using the **Datasets** menu item **Preceding Dataset** (Ctrl+Down) or clicking the **Previous** button in the toolbar.

Gotolibrary

Form:

```
Gotolibrary(BSTR Library, BSTR DataSet, BSTR Category)
```

Inputs:

`Library:` Library name.

`DataSet:` Dataset name.

`Category:` Category name.

Go to the specified dataset. If the string `Library` is empty, then the function will go to the dataset within the current library. If the strings `DataSet` and `Category` are both empty, then the dataset and category will go to the latest viewed dataset in the specified library.

ImportParsfile

Form:

```
ImportParsfile(BSTR FileName, short mode)
```

Inputs:

`Filename:` Name of Parsfile to import.

`mode:` Mode for handling datasets that already exist; 0 – do not import duplicated datasets, 1 – overwrite duplicated datasets.

Import an expanded Parsfile (any type). Applying this function is the same as using the File menu item **Import Parsfile (Any Export Type)**.

LaunchPlot

Form:

```
LaunchPlot(VOID)
```

Launch the plotter tool within VS Visualizer to view results for the current Run Control dataset.

Lock

Form:

```
Lock(VOID)
```

Lock the current dataset.

LockAll

Form:

```
LockAll(VOID)
```

Lock every dataset.

Minimize

Form:

```
Minimize (VOID)
```

Minimize the window of the VS Browser.

> **Note** After this function was provided years ago, changes were made in the VS Browser that cause the VS Browser to be forced into view when common actions are taken involving navigation (going to a specified dataset, trying to delete a dataset, going to a different library, etc.). The overall result is that the `Minimize` function is not as useful as might be expected for many applications.

Run

Form:

```
Run(BSTR DataSet, BSTR Category)
```

Inputs:

`DataSet:` Dataset name.

`Category:` Category name.

Run a simulation for the given dataset name and category name in the **Run Control** library. If `DataSet` and `Category` are empty, the currently active dataset is used.

Run_Background

Form:

```
Run_Background(VOID)
```

Similar to `Run` command above but run a simulation in background mode. It is recommended to be in the current Run dataset screen when executing this command. If it is not in the current Run dataset screen, it will go HOME and run a simulation.

Run_CheckError

Form:

```
Run_CheckError(BSTR DataSet, BSTR Category, [out] VARIANT_BOOL *
ThrewError)
```

Inputs:

`DataSet:` Dataset name.

`Category:` Category name.

Output:

`ThrewError:` Flag indicating whether the batch run terminated with an error.

Run a simulation for the given dataset name and category name in the **Run Control** library. This function is equivalent to the *Run* function above but with an output flag, `ThrewError`,

indicating whether the run terminated with an error. If `ThrewError` is true, call the *GetError* function to retrieve detailed error information.

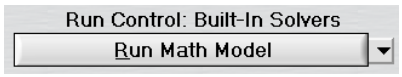
RunButtonClick

Form:

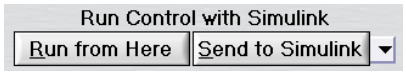
```
RunButtonClick(short iButton)
```

Input:

`iButton:` (iButton)th run button clicked, this can have values of 1, 2 or 3.



`iButton` = 1



`iButton` = 1 for left button, 2 for right button

`iButton` = 1 for left, 2 for center, 3 for right

This function simulates a mouse click on the button. It is used for setting up and running simulations for various options involving external models (e.g., Simulink).

Run_dSPACE

Form:

```
Run_dSPACE(BSTR DataSet, BSTR Category, short EntryPoint)
```

Inputs:

`DataSet:` Dataset name.

`Category:` Category name.

`EntryPoint:` Options involving a dSPACE simulation:

0: download vehicle model program, Parsfile and Run.

1: download Parsfile and Run.

3: download vehicle model program only.

4: launch dSPACE ControlDesk with project file(CDX file).

Run a simulation for the given dataset name and category name on a dSPACE target platform. If both `DataSet` and `Category` are empty, the currently active dataset is used.

Note In the past, EntryPoint option, "2: download Parsfile only was provided." However, it caused a memory leak and this option was discontinued.

Run_dSPACE_Background()

Form:

```
Run_dSPACE_Background(BSTR DataSet, BSTR Category, short
      EntryPoint)
```

Inputs:

`DataSet`: Dataset name.

`Category`: Category name.

`EntryPoint`: Options involving a dSPACE simulation:

0: download vehicle model program, Parsfile and Run.

1: download Parsfile and Run.

3: download vehicle model program only.

4: launch dSPACE ControlDesk with project file(CDX file).

Similar to `Run_dSPACE` above but it runs a program in background. If string `DataSet` and `Category` are empty, the currently active dataset is used. In this mode, GUI can be operated by a mouse or by COM.

Set_RundSPACE

Form:

```
Set_RundSPACE(short RunMode)
```

Inputs:

`RunMode`: Option to control a simulation.

0: stop

1: pause

2: play

While running Run_dSPACE, the simulation can be controlled.

Run_SCALEXIO

Form:

```
Run_SCALEXIO(BSTR DataSet, BSTR Category, short EntryPoint)
```

Inputs:

`DataSet`: Dataset name.

`Category`: Category name.

`EntryPoint`: Options involving a dSPACE simulation:

0: download Parsfile, launch ControlDeskNG and Run model.

1: download Parsfile to SCALEXIO target.

2: receive simulation result files from SCALEXIO.

3: build Simulink model by ConfigurationDesk.

4: launch dSPACE ControlDeskNG with project file.

Run a simulation for the given dataset (identified by name and category) on a dSPACE SCALEXIO target machine. If string `DataSet` and `Category` are empty, the currently active dataset is used.

CreateAllParFile

Form:

```
CreateAllParFile(BSTR DataSet, BSTR Category)
```

Inputs:

`DataSet:` Dataset name.

`Category:` Category name.

Create a run_all.par file for the given dataset name and category name in the **Run Control** library. If `DataSet` and `Category` are empty, the currently active dataset is used.

StartAnimator

Form:

```
StartAnimator(VOID)
```

Launch the VS animator program with the current Run Control dataset for playback.

StartAnimatorAndPlot

Form:

```
StartAnimatorAndPlot(VOID)
```

Launch the VS animator program and plotter tool with the current Run Control dataset for playback.

StartLiveVideo

Form:

```
StartLiveVideo(VOID)
```

Launch the VS animator program for live play using the current Run Control dataset. This is done for real-time and driving-simulator applications.

CreateAnimatorParFile

Form:

```
CreateAnimatorParFile(VOID)
```

Create a animator.par file for the currently selected dataset.

Unlock

Form:

```
Unlock(VOID)
```

Unlock the currently active dataset.

UnlockAll

Form:

```
UnlockAll(VOID)
```

Unlock every dataset.

Unminimize

Form:

```
Unminimize (VOID)
```

If the VS Browser window is minimized, restore it. If the window is not minimized, then this function has no effect.

ReadGeoLocations

Form:

```
ReadGeoLocations (BSTR fileName)
```

Inputs:

`fileName:` Name and location of a GPS input file.

Calling `ReadGeoLocations` will result in GPS coordinate data to be imported to the **Path: X-Y** screen. This COM function will only operate correctly when the user has navigated to the **Path: X-Y** screen and unlocked the current dataset.

UpdateRoad3DShape

Form:

```
UpdateRoad3DShape (VOID)
```

Calling `UpdateRoad3DShape` will update the road surface 3D shape files for the dataset currently viewed on the **Road: 3D Surface** screen. This COM function will only operate correctly when the user has navigated to the **Road: 3D Surface** screen and unlocked the current dataset.

SetOverlayPlotFile

Form:

```
SetOverlayPlotFile(BSTR BlueID, BSTR FileName)
```

Inputs:

`BlueID`: Keyword for a data link. All keywords have the form: `#BlueLink`*ID*, where *ID* goes from 0 to *n-1* (*n* is the number of links on the screen). You can obtain this keyword interactively by right-clicking on the link of interest.

`FileName`: Name of the file to use for the plot overlay. The file must exist.

Calling `SetOverlayPlotFile` will set the input file for the plot overlay on the current run. The file can be any valid CSV, ERD, or VS file produced by another run. This COM function will only operate when the user is on the **Run Control** screen and has unlocked the current dataset.

RewriteReindex

Form:

```
RewriteReindex(VOID)
```

Calling `RewriteReindex` will perform a full database rewrite and reindex.

ReindexDatabase

Form:

```
ReindexDatabase(VOID)
```

Calling `ReindexDatabase` will perform a full database reindex.

RewriteDatabase

Form:

```
RewriteDatabase(VOID)
```

Calling `RewriteDatabase` will perform a full database rewrite.

RewriteAllRuns

Form:

```
RewriteAllRuns(VOID)
```

Calling `RewriteAllRuns` will rewrite all run_all.par files in the Run Control library.

<This page is intentionally blank>

Mechanical Simulation

755 Phoenix Drive, Ann Arbor MI, 48108, USA
Phone: 734 668-2930 • Fax: 734 668-2877 • Email: info@carsim.com
carsim.com