

## Using Microsoft Visual Basic to Create a Graphical Front-End for a CSIM19 Model

This document describes how to use Microsoft Visual Basic to quickly and easily create a graphical front-end for a CSIM19 simulation written in C. The example used in this document is a simple service center phone bank. The source for this example is included with this document. You will not be able to compile the source yourself unless you own the CSIM19 libraries. This document assumes that readers are reasonably familiar with CSIM19 and basically proficient in Microsoft Visual Basic and Microsoft Visual C++. Obviously the methods described here are also applicable in other development environments, as long as users know their preferred development tools well enough.

The first step in this process is to create your CSIM model as you would normally. In our example, the model was first written and compiled as a standard Win32 console application using Microsoft Visual C++ 6. The original console program is included in this tutorial package and is called `phone.exe`. In this stage it is easy to find and fix all the bugs in the model itself without worrying about problems anywhere else. It is important to verify that the model gives the correct output at this stage, so that it will be easier to catch any bugs in the GUI later.

At this point we can design the Visual Basic interface. For our simple example, the program only really needs one, single form. In the source project there are two forms, but one is simply a home-made help screen. All the user needs to do is give the model some numbers and then get some other numbers back. For this, the Visual Basic form just needs a text field for each simulation parameter, and a space with some label objects to show the results. A simple mschart object has also been included in the example to show the results of several trials when the user enters a range for the number of operators to try.

Now that the two parts of the application are ready, (the simulator and the interface), it is time to re-build the model as a library file. First, we should remove any statements such as `fprint`, `cin`, `cout`, or anything else that uses the standard I/O stream. Obviously, none of these functions would make sense in a dynamic library. Next, we compile the program as a DLL. To do this in Microsoft Visual C++ 6, start a new project as a Win32 dynamic-link library, and add the source files to the project.

To get parameters from the interface to the model in the DLL, we call the DLL with a number of parameters, and we write the C code to handle those parameters. In this case, the function we will be calling to run the simulation is called `sim`. This is CSIM19's `main`, so we simply define `sim` to take parameters when it is called. When the user presses the "Run" button on the interface, the Visual Basic program first checks for invalid values in any of the fields, then it takes the values from the text box objects on the form and stores them to separate variables. The program then calls the library with the appropriate variables in the function call, and the library receives the parameters from the function call.

At this point, the simulator can get numbers in, but it has no way to get results back out. There are a number of ways this could be done, and for different models different methods may work better. For our simple example, since there are only a handful of numbers to get out of the model, we can just define a different function in the library code for each output value. In the DLL code, there are a number of functions which do nothing except return values to their callers. After the simulation runs, the Visual Basic program calls each of these functions in

sequence, the appropriate values are passed up the call stack, and the program stores the values as variables and prints them on the screen in label boxes.

Now, assuming there are no bugs, we can compile the library and the Visual Basic project and test them. Once the basics are taken care of, other things can be added for an easier user experience. In our example, we added the ability to run the simulation many times with different values for the number of operators and see the results as a graph. This was a simple task. The Visual Basic program contains a loop to increment the value and call the library each time for the different inputs. Then, a few more loops take care of adding the resulting values from an array to the `mschart` object.

Now the simulator is ready to go, complete with an easy interface. Obviously there are plenty of applications for this idea. While this method cannot provide anything as complicated as 3D renders of simulations, it does help to show that CSIM19 is not limited to white text floating on a black background. Please remember that Mesquite cannot support Visual Basic or anything else, except CSIM itself. This document is intended only to help our users get started on a project. For a more detailed notion of how to call library routines from Basic programs, examine the source code for the included example, or consult Microsoft's free Visual Basic reference materials at <http://msdn.microsoft.com/library>. Please feel free to contact Mesquite Software for support information regarding CSIM.

Mesquite Software, Inc.  
8500 N. Mopac Expwy, Suite 825  
Austin, Texas 78759  
In the US: (800) 538-9153  
<http://www.mesquite.com>

Notes on using the included example code –

If you obtained this document from your source media for a purchased copy of CSIM, the source files are in a directory called `VB Tutorial`, under the `Documentation` directory of your CSIM installation. If, for example, you installed CSIM to `C:\CSIM`, then the example files would be under `C:\CSIM\Documentation\VB Tutorial`.

In the `VB Tutorial` folder there are four files and two directories.

- `Phone.exe`: the simulation model built as a console application
- `Phone Demo.exe`: the application compiled from the Visual Basic project
- `Phonedll.dll`: the library containing the simulation routines, compiled from the C++ source
- `Tech_notes.txt`: a few quick technical notes on the example project
- `C++ Source`: directory containing the C++ source files used to compile `phonedll.dll`
- `Visual Basic Source`: directory containing the Visual Basic project and forms used to create `Phone Demo.exe`