

---

Version 8.0

# **VisSim/C-Code User's Guide**

**By Visual Solutions, Inc.**

---

**Visual Solutions, Inc.**

**VisSim/C-Code User's Guide Version 8.0**

Copyright	© 2010 Visual Solutions, Inc. All rights reserved.	Visual Solutions, Inc. 487 Groton Road Westford, MA 01886
Trademarks	VisSim, VisSim/Analyze, VisSim/CAN, VisSim/C-Code, VisSim/C-Code Support Library source, VisSim/Comm, VisSim/Comm C-Code, VisSim/Comm Red Rapids, VisSim/Comm Turbo Codes, VisSim/Comm Wireless LAN, VisSim/Fixed-Point, VisSim/Knobs & Gauges, VisSim/Model-Wizard, VisSim/Motion, VisSim/Neural-Net, VisSim/OPC, VisSim/OptimizePRO, VisSim/Real-TimePRO, VisSim/State Charts, VisSim/Serial, VisSim/UDP, VisSim Viewer, and flexWires are trademarks of Visual Solutions. All other products mentioned in this manual are trademarks or registered trademarks of their respective manufacturers.	
Copyright and use restrictions	The information in this manual is subject to change without notice and does not represent a commitment by Visual Solutions. Visual Solutions does not assume any responsibility for errors that may appear in this document.  No part of this manual may be reprinted or reproduced or utilized in any form or by any electronic, mechanical, or other means without permission in writing from Visual Solutions. The Software may not be copied or reproduced in any form, except as stated in the terms of the Software License Agreement.	
Acknowledgements	The following engineers contributed significantly to preparation of this manual: Mike Borrello, Allan Corbeil, and Richard Kolk.	

---

# Contents

<b>Introduction</b>	<b>1</b>
What is VisSim/C-Code .....	1
Are you planning to port your C code to another platform .....	1
What you need in order to use VisSim/C-Code .....	1
The VisSim product family .....	2
Resources for learning VisSim/C-Code.....	4
Interactive webinars.....	4
VisSim movies .....	4
Sample diagrams .....	4
Training .....	4
Additional reading material.....	5
<b>Preparing for Code Generation</b>	<b>7</b>
Basic preparation.....	7
Checking for incomplete wiring.....	7
Removing unsupported blocks.....	8
Resetting the integration algorithm.....	8
Preparing for DLL or simulation object generation.....	8
Generating code from automatically-generated DLLs .....	9
Translation of VisSim variable names.....	9
<b>Creating a Stand-Alone Executable</b>	<b>11</b>
Generating an executable file .....	11
Running an executable program .....	12
Output data.....	12
Exporting data.....	13
<b>Automatic DLL Generation</b>	<b>15</b>
What you can do with DLLs.....	15
Creating a DLL.....	15
Calling a DLL from a VisSim diagram .....	17
Verifying DLL results .....	18
Comparing simulation speed .....	18
Building a custom DLL.....	19
Troubleshooting .....	20
<b>Simulation Object Generation</b>	<b>23</b>
Creating a simulation object.....	23
Communicating with an embedded simulation object.....	24
Using the createSim function .....	25
Using the vsmCgRuntimeCommand.....	25
Using the vsmCgGetLastErrorString().....	26

Sample file with simulation object interface .....	26
<b>Generating Source Code</b>	<b>29</b>
Generating C code .....	29
Examining a .C file .....	30
<b>VisSim/C-Code Support Libraries</b>	<b>31</b>
Object files .....	31
<b>Targeting C Code for Unsupported Platforms</b>	<b>33</b>
C support library source code .....	33
Copying the C support library source code to your hard disk .....	34
Compiling and linking the C support library source code .....	34
<b>Installing VisSim/C-Code</b>	<b>35</b>
Installation requirements .....	35
Installing VisSim/C-Code .....	35
<b>Index</b>	<b>37</b>

# Introduction

This section contains...

---

## What is VisSim/C-Code

VisSim/C-Code provides an efficient way to automatically:

- Translate an entire VisSim diagram in a stand-alone executable file
- Create a VisSim-callable DLL
- Generate customizable C code
- Generate a simulation object that can be embedded into a custom application

Because the C code generated by VisSim/C-Code is optimized for speed, the resulting executables and DLLs will run up to five times faster than their block diagram counterparts. This is particularly useful if your applications have high sampling rates (typically less than 1 ms).

### Are you planning to port your C code to another platform

The C code generated by VisSim/C-Code can be ported to other platforms for compilation and linking, provided you have an ANSI C compiler and C support library for that platform. For the latest list of platform-specific C support libraries, contact Technical Support.

The generated C code can also run on embedded controllers or DSP chips provided you have the VisSim/C-Code Support Library Source Code.

### What you need in order to use VisSim/C-Code

VisSim/C-Code is an extension to Professional VisSim; to use VisSim/C-Code, Professional VisSim must be installed on your computer.

You also need an ANSI C compiler. It is recommended, though not required, that you use the Microsoft Visual C6.x or higher compiler; VisSim/C-Code is configured to use these compilers. If you choose to use a different compiler, refer to the documentation that accompanies the compiler for information on compiling and linking your source code.

---

# The VisSim product family

The VisSim product family includes several base products and product suites, as well as a comprehensive set of targeted add-on modules that address specific problems in areas such as data communications, data acquisition, linearization and analysis, and digital signal processing.

## Base products and product suites

Product	Function
Professional VisSim	<p>Model-based design, simulation, testing, and validation of dynamic systems.</p> <p>A personal version, VisSim PE, is also available. VisSim PE limits diagram size to 100 blocks.</p>
VisSim/Comm Suite	<p>Simulates end-to-end communication systems at the signal level using 200+ communications, signal processing, and RF blocks.</p> <p>Includes Professional VisSim and VisSim/Comm blockset.</p> <p>A personal version, VisSim/Comm Suite PE, is also available. VisSim/Comm PE limits diagram size to 100 blocks and limits the Communication blockset. See the VisSim/Comm datasheet for details.</p> <p>VisSim/Comm Suite add-on modules are available for real-time data acquisition (Red Rapids digital tuner card); modeling PCCC turbo codes, including UMTS specification; and for support of Bluetooth, 802.11 a/b/g (Wi-Fi), and ultrawideband wireless designs.</p>
VisSim/Embedded Controls Developer Suite	<p>Rapidly prototypes and creates embedded controls for DSPs, DSCs, and MSP430 microcontrollers. You can simulate and generate scaled, fixed-point ANSI C code, as well as code for on-chip peripherals.</p> <p>Includes Professional VisSim, VisSim/C-Code, VisSim/Fixed-Point, and one user-specified target support.</p> <p>A personal version, VisSim/Embedded Controls Developer PE, is also available. VisSim/Embedded Controls Developer PE limits diagram size to 100.</p>
VisSim Viewer (free)	<p>Lets you share VisSim models with colleagues and clients not licensed to use VisSim.</p>

## Add-on modules

Add-On Module	Function
VisSim/Analyze	Performs frequency domain analysis of a linearized nonlinear subsystem.
VisSim/CAN	Interfaces with a USB CAN device to read and write CAN messages on the CAN bus.
VisSim/C-Code	Generates highly-optimized, ANSI C code that can be compiled and run on any platform that supports an ANSI C compiler.
VisSim/C-Code Support Library Source	Provides source code for the Support Library.
VisSim/Comm blockset	<p>Simulates end-to-end communication systems at the signal level using 200+ communications, signal processing, and RF blocks.</p> <p>A personal version, VisSim/Comm PE, is also available. VisSim/Comm PE is a subset of the Communication blockset. See the VisSim/Comm datasheet for details</p> <p>You can purchase VisSim/Comm add-on modules for real-time data acquisition (Red Rapids digital tuner cards); for modeling PCCC turbo codes, including UMTS specification; for support of Bluetooth, 802.11 a/b/g (Wi-Fi), and ultrawideband wireless designs.</p>
VisSim/Fixed-Point	Simulates the behavior of fixed-point algorithms prior to code generation and implementation of the algorithm on the fixed-point target.
VisSim/Knobs and Gauges	Provides dynamic gauges, meters, and knobs for process control, and measurement and validation systems.
VisSim/Model-Wizard	Generates transfer function model from historic or real-time data.
VisSim/Motion	Simulates motor control systems with customizable amplifiers, controllers, filters, motors, sensors, sources, tools, and transforms.
VisSim/Neural-Networks	Performs nonlinear system identification, problem diagnosis, decision-making prediction, and other problems where pattern recognition is important.
VisSim/OPC	Connects to any OPC server and log data or run a virtual plant in VisSim for offline tuning.
VisSim/OptimizePRO	Performs generalized reduced gradient method of parameter optimization.
VisSim/Real-TimePRO	Performs real-time data acquisition and signal generation using I/O cards, PLCs, and DCSs.
VisSim/Serial	Performs serial I/O with other computers.
VisSim/State Charts	Creates, edits, and executes event-based systems.
VisSim/UDP	Performs data exchange over the internet using UDP.
VisSim Viewer (free)	Lets you share VisSim models with colleagues and clients not licensed to use VisSim.

---

## Resources for learning VisSim/C-Code

For those of you that are new to VisSim, we have provided several free services to make your transition to VisSim fast, smooth, and easy:

- [Interactive webinars](#)
- [VisSim movies](#)
- [Sample diagrams](#)

---

## Interactive webinars

Interactive webinars offer you the opportunity to meet with Visual Solutions product specialists who will introduce and demonstrate our software products live on your computer and answer any questions you have. Each webinar is approximately 45 minutes long. To learn more about our interactive webinars, go to

<http://www.vissim.com/webinars/webinars.html>.

---

## VisSim movies

Designed by Visual Solutions application engineers, the VisSim movies guide you through the creation, simulation, debugging, and optimizing of block diagrams that cover a broad range of engineering disciplines.

You can access the movies from your VisSim CD under the \MOVIES directory. Or, you can go to [http://www.vissim.com/support/vissim\\_instructional\\_movies.html](http://www.vissim.com/support/vissim_instructional_movies.html) and download the movies to your computer.

---

## Sample diagrams

VisSim 8.0 includes a directory of fully documented sample diagrams. These diagrams illustrate both simple and complex models spanning a broad range of engineering disciplines, including aerospace, biophysics, chemical engineering, control design, dynamic systems, electromechanical systems, environmental systems, HVAC, motion control, process control, and signal processing.

### To access sample diagrams

Click on the **Diagrams** menu in VisSim.

Click on **Examples > Applications**.

---

## Training

Visual Solutions offers training sessions for learning and gaining expertise in VisSim and the VisSim family of add-on products. Training sessions are conducted at Visual Solutions training facility in Westford, MA, as well as at customer sites and as online webinars.

For information on setting up a training session, contact [sales@vissol.com](mailto:sales@vissol.com).

---

## Additional reading material

Though familiarity with the C programming language is not necessary to use VisSim/C-Code and run executable programs, if you want to manipulate the generated C code, you should be able to program in C. The C compiler you have chosen comes with documentation; however, for a full-length description of the C programming language and software engineering principles of program construction, we recommend *C: A Software Engineering Approach* (P. Darnell and P. Margolis, Springer-Verlag).



# Preparing for Code Generation

This section contains...

---

## Basic preparation


Whether your goal is to generate a stand-alone executable file, a DLL function, or source C code, you should perform the following tasks:

- Check your block diagram for incomplete wiring
- Remove unsupported blocks from your block diagram
- Set the integration algorithm to Runge Kutta 2<sup>nd</sup>

### Checking for incomplete wiring

A block diagram containing one or more unconnected inputs will probably produce a faulty .C file. An easy way to verify that all connector tabs are wired is to use the Check Connections option before you generate the C code.

#### To check for incomplete wiring

1. Choose **Simulate > Simulation Properties**.
2. Click on the **Preferences** tab.
3. Activate the **Check Connections** option and click on **OK**.
4. Choose **Simulate > Go** or press the  toolbar button.
5. For each unconnected connector tab, VisSim highlights the block in red and displays a dialog box indicating the name of the faulty block and the unconnected input. The dialog box provides the following options:
  - **Abort or Retry.** VisSim finishes checking the diagram for incomplete wiring, then halts the simulation.
  - **Ignore.** VisSim finishes checking the diagram for incomplete wiring, then continues the simulation.

Blocks remain highlighted in red until you click the right mouse button over them, or choose the **Edit > Clear Errors** command, which clears all highlighted blocks.

## Removing unsupported blocks

A small set of blocks is not supported by VisSim/C-Code. When VisSim/C-Code encounters one of these blocks, it either translates the block into an ASCII data stream or a call to the EMPTY function.

Most of VisSim's Signal Consumer blocks are translated into function calls that produce ASCII data streams. ASCII data streams can be redirected to a file (after compilation and linking is complete) and then read into any number of applications with graphical plotting capabilities.

For maximum performance of your executable or DLL, it is recommended to remove unsupported blocks.

The table below lists the blocks not supported by VisSim/C-Code.

Block name	Produce ASCII stream	Call EMPTY function
animate		✓
bezel		✓
constraint		✓
cost		✓
DDE, DDEreceive, DDEsend		✓
display	✓	
globalConstraint		✓
histogram		✓
light	✓	
lineDraw		✓
meter	✓	
neuralNet		✓
parameterUnknown		✓
plot	✓	
rt-DataIn*	✓	
rt-DataOut*	✓	
stripChart	✓	
unknown		✓

\* Call Technical Support for availability.

## Resetting the integration algorithm

The integration algorithm is re-set to Runge Kutta 2nd order if it was previously set to an algorithm other than Runge Kutta 4<sup>th</sup> order, Runge Kutta 2nd order, or Euler. VisSim/C-Code notifies you if it changes the integration algorithm.

---

## Preparing for DLL or simulation object generation

Before you can generate a DLL or a simulation object, you must:

- Encapsulate the blocks to be converted into a DLL or simulation object in a single compound block.
- Make sure the inputs and outputs on the compound block correspond with the inputs and outputs on the DLL or simulation object. Note that for DLL generation, connector labels are carried over to the DLL making it easier to wire the DLL into the diagram. For simulation object generation, the connector labels are available in a data structure in the generated C file.

#### To create a compound block

1. Select the blocks that are to be converted into a DLL.
2. Choose **Edit > Create Compound Block** to create a compound block.
3. Label the input and output connector tabs of the compound block by double-clicking the mouse over each connector tab and entering a unique name in the Connector box.

---

## Generating code from automatically-generated DLLs

You can include pre-existing, user-written or automatically-generated DLLs in the portion of the diagram from which you are generating code. To do so, you must declare the DLL functions in USERDLL.H and include the library for the DLL in VSMDLL32.BAT.

For example, to generate a DLL from a compound block in which the DLL named READ\_INPUT\_FILE is embedded, do the following:

- In USERDLL.H, add

```
__declspec(dllexport) void __stdcall EXPORT READ_INPUT_FILE (double p[],double in [],double out[]);
```

This line declares the exported DLL function READ\_INPUT\_FILE so that automatic DLL code generation will make the proper external reference to it.

- In VSMDLL32.BAT, add:

```
set userlibs=READ_INPUT_FILE.LIB
```

This line associates the shell variable userlibs with the library file for the exported DLL function.

- To associate multiple libraries with userlibs, separate each library file with an empty space. For example:

```
set userlibs=READ_INPUT_FILE.LIB READ_OUTPUT_FILE.LIB
```

**Note:** If the user-written file is a .CPP file, you must prefix the exported functions in the .CPP file with extern "C". For example:

```
extern "C" __declspec(dllexport) void __stdcall EXPORT READ_INPUT_FILE (double p[],double in [],double out[]);
```

This line causes the external name generated by the .CPP file to be compatible with VisSim DLL naming conventions.

---

## Translation of VisSim variable names

The code generator tries to retain VisSim variable names in the generated code. If, however, a variable name contains +, -, \*, #, @, ! they are converted into underscore characters. This limitation is important when naming blocks that will eventually be compiled into C code; that is, you must avoid names that differ only in the above punctuation characters.



# Creating a Stand-Alone Executable

This section contains...

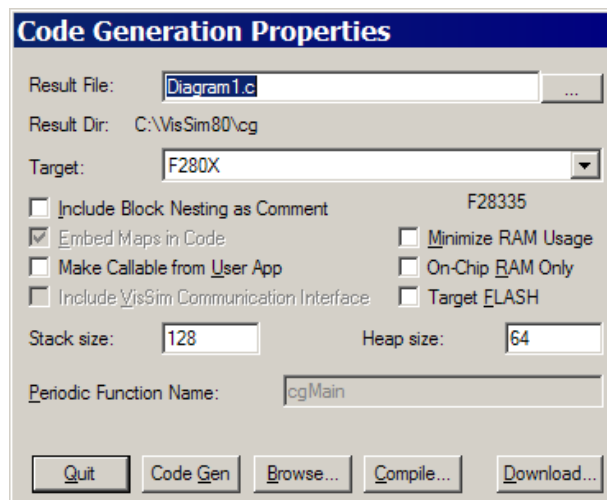
---

## Generating an executable file

This procedure describes how to create a stand-alone executable file from a block diagram. Before you begin, prepare the diagram for code generation.

### To generate a stand-alone executable file

1. Open the block diagram to be converted into an executable file.
2. Choose **Tools > Code Gen**. The Code Generation Properties dialog box appears.



3. The Result File box displays *diagram-name.C*, where *diagram-name* is the name of the current block diagram. By default, VisSim/C-Code uses the block diagram name as the name for the stand-alone executable. For example, if you enter ACMOTOR.C, VisSim/C-Code creates an executable file called ACMOTOR.EXE.
4. The Result Dir box indicates where the executable file will be stored. For convenience, the destination directory should be the directory that contains the C support library (CG32.LIB). To change the directory, click on the ... button and select a new directory.

5. The Target box contains the target platform for code generation. Choose the Host option, if it is not already selected.
6. Choose from the following options:

Activate this option	To
Include Block Nesting as Comment	Include comments in the generated code that indicate the compound blocks that correspond to the code.
Embed Maps in Code	Insert map file contents directly into the generated code. When this option is activated, the resulting executable will be portable because the map file is no longer needed.
Make Callable from User App	Generates the C code in such a way as to be callable from a user application or RTOS.
Minimize RAM Usage	Uses smallest amount of RAM but does not allow numerical integration. Turn this option ON if you are using numerical integration.

**Code generation options that do not apply to stand-alone executables**

**When creating a stand-alone executable, the Label Block and Connectors, Block NameFunction Name, Target FLASH, Stack Size, and Heap Size options do not apply and should therefore be de-activated.**

7. Click on the **Compile** button.
8. VisSim/C-Code opens a text window in which it displays the creation of the executable file. When the file has been generated, press any key to return to the Code Generation Properties dialog box.
9. Click on the **Done** button.

---

## Running an executable program

To run an executable program, you enter the executable file name at the MS/DOS command prompt or you can enter the file name in the text box for the Run command in the Start menu.

### Output data

The output information produced by Signal Consumer blocks appear as ASCII data streams. For example, the results of a single input `plot` block are displayed in a single column. Each row reflects the signal value at each step in the simulation. The number of rows equals the total number of steps in the simulation.

If a Signal Consumer block has multiple inputs, the results for each input signal are displayed in separate columns.

## Exporting data

If you want another program to analyze or manipulate your simulation data, you can wire an `export` block into the diagram before creating the executable file. The `export` block writes from one to 32 signals to a file in `.DAT`, `.M`, `.MAT`, or `.WAV` file format. For more information on the `export` block, see the *VisSim User's Guide*.



# Automatic DLL Generation

This section contains...

---

## What you can do with DLLs

- Speed up simulation time  
When a block diagram contains DLLs, it requires less disk space and memory since its executable program files contain the names of the DLL functions but not the code for the functions. For particularly large diagrams, the use of VisSim-callable DLLs can significantly increase the speed of your simulations.
- Perform multi-rate execution  
When a compound block is converted into a DLL, the DLL retains the step size and integration method in use at the time of DLL generation. If the diagram that calls the DLL has a faster or slower step rate, the DLL skips or adds steps to maintain its own clock rate. This allows you to have a low frequency overall diagram with high frequency components compiled as DLLs.
- Protect intellectual property  
Because DLLs cannot be reverse-engineered into readable source code, you can be sure that no one can access your intellectual property.
- Interface with other applications or simulators

---

## Creating a DLL

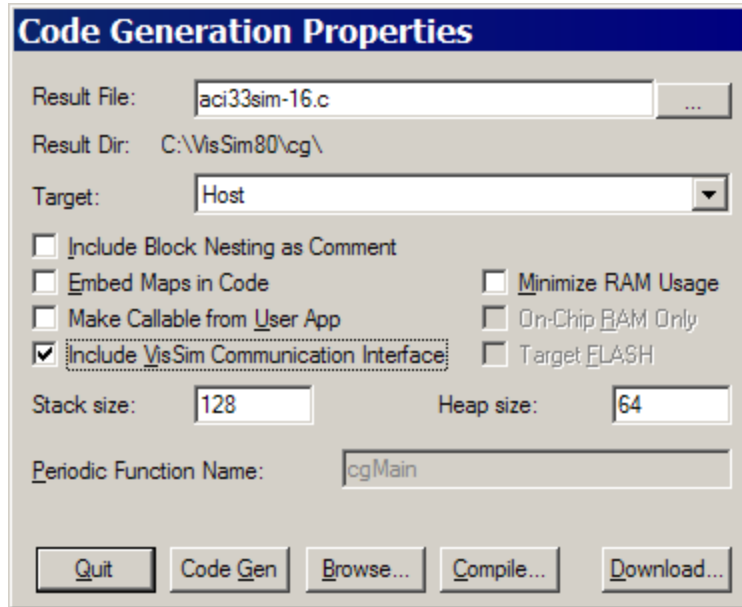
Only compound blocks can be converted into DLLs. When a compound block is converted into a DLL, the DLL retains the step size and the integration method in use at the time of DLL generation, and not those selected or specified by the diagram calling the DLL. The DLL does, however, use the simulation start and end times of the diagram that calls it.

The generated DLL includes the block and connector names. When the resulting DLL has numerous input and output connector tabs, the connector names make it easy to identify the correct wiring paths

### To create a DLL

1. Open the block diagram that contains the blocks you want to convert into a DLL.

2. Prepare the diagram for DLL generation.
3. Select the compound block to be converted into a DLL.
4. Choose **Tools > Code Gen**.
5. The Code Generation Properties dialog box appears.



The Result File box displays *diagram-name.C*, where *diagram-name* is the name of the current block diagram. By default, VisSim/C-Code uses the block diagram name as the name for the DLL. For example, if you enter ACMOTOR.C, VisSim/C-Code creates a DLL called ACMOTOR.DLL.

If you are creating more than one DLL from a single VisSim diagram, be sure to give each DLL a unique name.

6. The Result Dir box indicates where the DLL will be stored. If you want to change the location, click on the ... button.
7. The Target box contains the target platform for code generation. Choose the **Host** option, if it is not already selected.
8. Activate the **Include VisSim Communication Interface** option. This option allows the generated code to include calls to send data to and receive data from VisSim. This option is dimmed when no compound block is selected.
9. Choose from the following options:

Activate this option	To
Include Block Nesting as Comment	Include comments in the generated code that indicate the compound blocks that correspond to the code.
Embed Maps in Code	Insert map file contents directly into the generated code. When this option is activated, the resulting DLL will be portable to platforms that do not support a file system, because the map file is no longer needed.
Make Callable from User App	Generates the C code in such a way as to be callable from a user application or RTOS..

Minimize RAM Usage	Uses smallest amount of RAM but does not allow numerical integration. Turn this option ON if you are using numerical integration.
Periodic Function Name	Specifies the name of the DLL function. It defaults to cgMain. There is no need to edit this name.

**Code generation options that do not apply to automatic DLL generation**

When automatically creating DLLs, the Target FLASH, Stack Size, and Heap Size options do not apply and should therefore be de-activated.

10. Click on the **Compile** button.
11. VisSim/C-Code opens a text window in which it displays DLL creation. When the DLL has been generated, press any key to return to the Code Generation Properties dialog box.
12. Click on the **Done** button.

---

## Calling a DLL from a VisSim diagram

The process of calling a DLL from a VisSim diagram involves binding the DLL to a userFunction block and then wiring the userFunction block into the diagram. During simulation, each time the userFunction block is executed, VisSim calls the DLL.

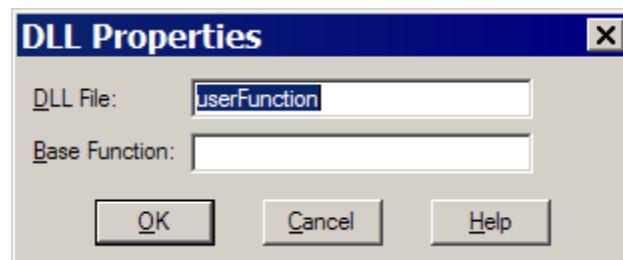
After you bind a DLL to a userFunction block, VisSim/C-Code renames the userFunction block with the DLL name. For example, if you created a DLL named AC Motor, VisSim/C-Code renames the userFunction block AC Motor.

If you elected to retain connector labels when you created the DLL, you can display the labels on the userFunction block using the View > Connector Labels command. Connector labels make it easy to correctly wire a userFunction block into a diagram.

### To bind a DLL to a userFunction block

1. From the **Blocks** menu, drag a **userFunction** block into your diagram.
2. Choose **Edit > Block Properties**.
3. Point to the **userFunction** block and click the mouse.

The DLL Properties dialog box appears.



4. Do the following:
  - In the DLL File box, enter the complete file specification of the DLL. This name corresponds with name specified in the Result File box in the Code Generation Properties dialog box. If you are not sure where the DLL file resides, click on the Select button to locate it.

- In the Base Function box, enter **cgMain**.
5. Click on the **OK** button, or press **ENTER**.

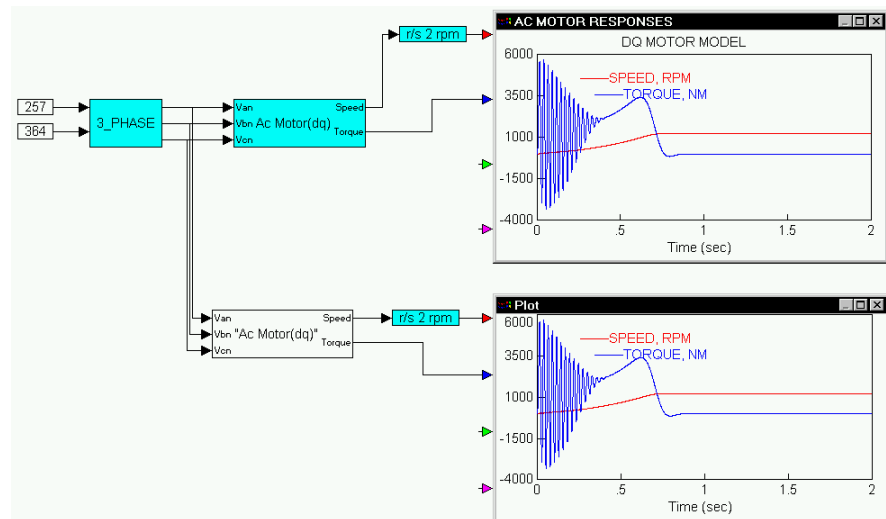
#### To view connector labels on a DLL

- Choose **View > Connector Labels**.

## Verifying DLL results

Before editing the block diagram to replace the existing blocks with a corresponding DLL, it is a good idea to verify that the DLL operates correctly.

For example, in following diagram, a DLL is created from the *AC Motor (dq)* compound block. To verify that the DLL operates correctly, the inputs to the compound block are fed into the DLL. A second plot block, with the same properties as the original plot is placed in the diagram. If both plots register the same results when you simulate the diagram, then the DLL is correct.



## Comparing simulation speed

You can easily compare how much faster a simulation runs with a DLL than without using the **Notify At Simulation End** option in the **Simulation Properties** dialog box. This option displays how long it takes to run a simulation in simulated time and real time.

#### To compare performance

1. Choose **Simulate > Simulation Properties**; then click on the **Preferences** tab.
2. Activate the **Notify At Simulation End** option and click on the **OK** button.
3. Disconnect the DLL from the diagram.
4. Run the simulation.
5. Reconnect the DLL to the diagram and disconnect the corresponding compound block from the diagram.
6. Run the simulation.
7. Compare the real time simulation results from the two simulation runs.

---

# Building a custom DLL

If you want to add a custom dialog box to a DLL, you have to compile and link the code manually.

Nowadays, most languages have a Project Build facility that automates the process of building an executable or DLL. The following procedures guide you through the process of building a project using the Microsoft Visual C++ v6 or higher compiler. Refer to the documentation for the application language you are using for specific instructions.

## To build a custom DLL with Microsoft Visual C++ v6

1. Invoke the Compiler environment.
2. Do the following to create a new project workspace:
  - Choose **File > New**.
  - Under the New category, select **Project Workspace** and click on **OK**.
  - In the New Project Workspace dialog box, do the following:
    - Under Project Type, select **Windows Dynamic Link library (.DLL)**.
    - In the Name box, enter a name for the project. This name becomes the name of the DLL.
    - If you want to change the directory in which the project is stored, enter a new location in the Location box.
    - Click on the **Create** button.
3. Choose **Insert > Files Into Project** to add the following files to the project workspace:
  - The generated .C file
  - \VISSIM80\CG\LIB\CGDLL32.LIB
  - \VISSIM80\VSDK\LIB\VISSIM32.LIB

**Note:** To specify file names, enter the full pathname, including drive specification for CGDLL32.LIB and VISSIM32.LIB.
4. Do the following to establish the settings for the build:
  - Choose **Build > Settings**.  
The Project Settings dialog box appears.
  - Do the following:
    - Click on the **C/C++** tab.
    - Under Category, select **Preprocessor**.
    - In the Additional Include Directories box, specify the following:  
\VISSIM80\VSDK\INCLUDE, \VISSIM80\CG\INCLUDE

**Note:** To specify directory names, enter the full pathname, including the drive specification for the INCLUDE directories.
  - Click on the **OK** button.
5. Choose **Build > Build** to build the project.

## To build a custom DLL with Microsoft Visual C++ v6

1. Invoke the Compiler environment.
2. Do the following to create a new project workspace:
  - Choose **File > New**.
  - Under the New category, select **Win32 Dynamic Link Library**.
  - To change the directory in which the project is stored, enter a new location in the Location box
  - In the Project Name box, enter a name for your project.
  - This becomes the subdirectory where your project is stored, as well as the name of your DLL.
  - Click on the **OK** button.
3. Choose **Project > Add To Project > Files** to add the following files to the project workspace:
  - Your custom .C file or VisSim-generated .C file
  - \VISSIM80\CG\LIB\CGDLL32.LIB
  - \VISSIM80\VSDK\LIB\VISSIM32.LIB

**Note:** To specify file names, enter the full pathname, including drive specification for CGDLL32.LIB and VISSIM32.LIB.
4. Do the following to establish the settings for the build:
  - Choose **Project > Settings**.  
The Project Settings dialog box appears.
  - Do the following:
    - Click on the **C/C++** tab.
    - Under Category, select **Preprocessor**.
    - In the Additional Include Directories box, specify the following:  
\VISSIM80\VSDK\INCLUDE, \VISSIM80\CG\INCLUDE

**Note:** To specify directory names, enter the full pathname, including the drive specification for the INCLUDE directories.
  - Click on the **OK** button.
5. Choose **Build > Build** to build the project.

## Troubleshooting

### ***Out of Environment Space error message***

The most common problem is an Out of Environment Space error message in the MS-DOS window. To rectify this problem, select the Memory tab in the MS-DOS Prompt Properties, and increase the Initial Environment setting to a larger value. Then try again.

### ***LINK warning about LIBCLIB***

If you receive a Link warning message during the build, you should instruct the Project Build facility to ignore LIBC.LIB. If you are using the Microsoft Visual C++ v6 compiler, follow these steps to remove LIBC.LIB:

1. Choose **Build > Settings**.
2. Under the Project Settings dialog box, click on the **Link** tab.
3. Under Categories, specify **Input**.
4. Under Ignore Libraries, enter **LIBC.LIB**.
5. Click on the **OK** button.
6. Choose **Build > Build**.

### ***LINK warning messages that can be ignored***

You may see additional link errors, such as LINK: warning LNK4099 : PDB vc40.pdb was not found with .\deb\FILEIO.OBJCgdl132.lib or at c:\src\cg\deb\vc40.pdb; linking object not found. These are compiler warnings that there is not sufficient information to create a debug database for the project.



# Simulation Object Generation

This section contains...

---

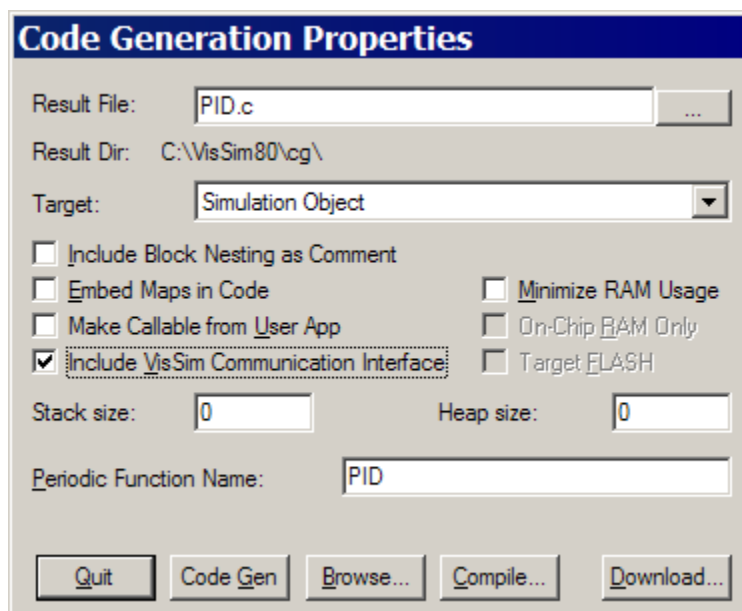
## Creating a simulation object

Only compound blocks can be converted into simulation objects. When a compound block is converted into a simulation object, it retains the step size and the integration method in use at the time it was generated, and not those selected or specified by the diagram in which the simulation object will be embedded. The simulation object does, however, use the simulation start and end times of the diagram that calls it.

### To create a simulation object

1. Open the block diagram that contains the blocks you want to convert into a simulation object.
2. Prepare the diagram for [simulation object generation](#).
3. Select the compound block to be converted into a simulation object.
4. Choose **Tools > Code Gen**.

The Code Generation Properties dialog box appears.



The Result File box displays *diagram-name.C*, where *diagram-name* is the name of the current block diagram. By default, VisSim/C-Code uses the block diagram name as the name for the generated .C file.

If you are creating more than one simulation object from a single VisSim diagram, be sure to give each simulation object a unique name.

5. The Result Dir box indicates where the simulation object will be stored. If you want to change the location, click on the ... button.
6. The Target box contains the target platform for code generation. Choose the **Simulation Object** option, if it is not already selected.
7. Activate the **Include VisSim Communication Interface** option; then choose from the following options:

Activate this option	To
Include Block Nesting as Comment	Include comments in the generated code that indicate the compound blocks that correspond to the code.
Activate this option	To
Embed Maps in Code	Insert map file contents directly into the generated code. When this option is activated, the resulting simulation object will be portable to platforms that do not support a file system, because the map file is no longer needed.
Make Callable from User App	Generate the C code in such a way as to be callable from a user application or RTOS.
Minimize RAM Usage	Uses smallest amount of RAM but does not allow numerical integration. Turn this option ON if you are using numerical integration.
Periodic Function Name	Specifies the name of the base simulation function. It defaults to cgMain. There is no need to edit this name.

***Code generation options that do not apply to simulation object generation***

When generating simulation objects, the Target FLASH, Stack Size, and Heap Size options do not apply and should therefore be de-activated.

8. Click on the **Compile** button.
9. VisSim/C-Code opens a text window in which it displays simulation object creation. When the simulation object has been generated, press any key to return to the Code Generation Properties dialog box.
10. Click on the **Done** button.

---

## Communicating with an embedded simulation object

VisSim/C-Code provides a support library that contains API functions to support code generation. These include, among other things, numerical integration, transfer function filter, time delay, and pulse train. The support library resides in the \CG\LIBRARY directory.

VisSim/C-Code also includes a sample file that invokes the API and shows how calls are used. This file, named SIMOBJ.C, also resides in \CG\LIBRARY.

### To communicate with your simulation object, follow these steps:

1. Open your custom application file.
2. Create a handle to your simulation object using the **createSim** function.
3. Make calls to the **vsmCgRuntime** command.

## Using the createSim function

The **createSim** function returns a handle to the simulation. With the handle, you can invoke **vsmCgRuntime** to run the simulation. Every time you call **createSim**, you get a unique handle to a new simulation object. This means that you can have multiple simulation objects embedded in your custom application.

```
variable = sim-name createSim();
```

The **sim-name** is the function name you provided when you generated the code.

The declaration for the **createSim** function is:

```
SIM_STATE *variable
```

## Using the vsmCgRuntimeCommand

This function runs the simulation using the specified input signals.

```
int vsmCgRuntimeCommand(  
    IN SIM_STATE * hSim,  
    IN int cmd,  
    IN double inSig[],  
    OUT double outSig[],  
    INOUT double * targetTime);
```

### Command values

**RTE\_RUN\_TO\_TIME** Runs the VisSim model until *\*targetTime* is reached. **InSig** contains a vector of double precision inputs to the simulation. Note that the order of the inputs is the top-down order of the original compound block connectors. On return, **outSig** contains the calculated output values.

### Return values for RTE\_RUN\_TO\_TIME

**VSM\_SIM\_MATH\_ERR** Indicates that a math error has occurred.

**VSM\_SIM\_END\_TIME\_EXCEEDED** Indicates that *\*targetTime* is past the current VisSim end time.

**VSM\_SIM\_USER\_STOP** Indicates that a user has pressed the stop button or that the stop block has been actuated.

**VSM\_ERR\_FILE\_ACCESS** A map or import file could not be opened.

**RTE\_SET\_EXIT\_CONDITION** Takes the string in **inSig** and sets it as a C expression to be evaluated at each time step for early simulation termination.

**RTE\_SET\_TIME** Moves VisSim time forward to *\*targetTime*. Note that blocks do not calculate during this call.

**RTE\_GET\_TASK\_TIME** Fills *\*targetTime* with the current VisSim time.

**RTE\_GET\_TIME\_STEP** Returns the current VisSim time step.

**RTE\_GET\_NEXT\_TIME** Returns the next time at which VisSim calculates new values.

## Using the `vsmCgGetLastErrorString()`

This function returns a text description of the most recent error. If there has not been an error, a NULL pointer is returned.

```
const char* vsmCgGetLastErrorString( IN SIM_STATE * hSim)
```

## Sample file with simulation object interface

The following .C file shows the interface to VisSim simulation object code generation. This is a generic wrapper for a single instantiation of a simulation object. You will replace this file with your own user interface that can instantiate any number of simulation objects.

```
#include <math.h>
#include <memory.h>
#include <stdlib.h>
#include "vsuser.h"
#include "cgen.h"

// Sample file to show how to create and interface to a simulation object
#include "vsmApp.h" // This file contains the vsmCgRuntimeEvent() commands
#include "cmdApi.h"
#define MAX_VSM_ARG 64
#define TIME_END 1

int main(int argc, char **argv)
{
    SIM_STATE *hSim; // Declare a handle to the sim
    double inSig[MAX_VSM_ARG], outSig[MAX_VSM_ARG], simTime,T,
    timeStep=.05;
    int a, retVal;
    double endTime=TIME_END*1.000000001;

    hSim = cgMainCreateSim(); // Create sim, return handle (Instantiate a simObject)

    vsmCgRuntimeCommand(hSim,RTE_GET_TIME_STEP,&timeStep,0,0); // Get
    sim timestep (optional)
    vsmCgRuntimeCommand(hSim,RTE_RESET,0,0,0); // Reset the sim
    (required before each run)
    for (T=0; T <= endTime; T+=timeStep)
```

```

{
    inSig[0] = 5; // To do: supply your arguments here
    inSig[1] = 5; // We supply some arg values here
    inSig[2] = 0;
    retVal =
vsmCgRuntimeCommand(hSim,RTE_RUN_TO_TIME,inSig,outSig,&T);
    if (retVal != VSM_SUCCESSFUL)
        {
            printf("vsmCgRuntimeCommand() returns '%s': ",
vsmCgGetLastErrorString(hSim));
            break; // Problem in sim, stop simulating
        }
        printf("T=%g:ST=%g: %g,%g\n",T,simTime, outSig[0],outSig[1]); // To do:
make use of VisSim results
    }
    return 0;
}

```



# Generating Source Code

This section contains...

---

## Generating C code

During code generation, VisSim/C-Code translates blocks into C source statements. The .C file typically includes the following information:

- Include directives to call the necessary header files
- Declaration of the variables
- A function where all the simulation calculations are performed
- A main function, which calls the above function and contains the simulation parameters, including the duration of the simulation, the simulation time step, and the integration method
- A void limitIntegOutput ( ) { } function to implement limitedIntegrator blocks (if needed)

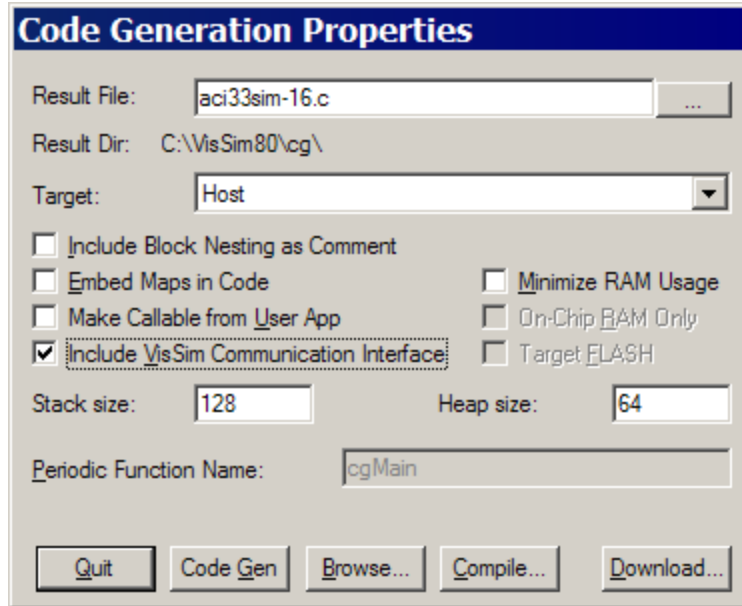
### Unsupported blocks

A [small set of blocks](#) are unsupported in VisSim/C-Code and are translated into function calls that either produce ASCII data streams or EMPTY returns. If you are familiar with the C language, you can write your own C functions for unsupported blocks.

### To generate a .C file

1. Prepare the block diagram for [source code generation](#).
2. Choose **Simulate > Code Gen**.

The Code Generation Setup dialog box appears.



3. In the Result File box, enter a name for the generated .C file. If you do not specify a file name, VisSim uses the current block diagram name and appends a .C extension.
4. The Result Dir box indicates where the .C file will be stored. For convenience, the destination directory should be the directory that contains the C support library (CG32.LIB). To change the directory, click on the Change Dir button and select a new directory.
5. The Target box contains the target platform for code generation. Choose the **Host** option, if it is not already selected.
6. Choose the additional code generation options you want.

If the source code is to be translated into a	See
Stand-alone executable	<a href="#">Creating a stand-alone executable.</a>
DLL	<a href="#">Creating a DLL.</a>

7. Click on the **Code Gen** button.

---

## Examining a .C file

You may find that you want to examine the source code you generate. The Browse feature allows you to examine or edit C source code.

On the Windows platforms, Browse starts up Microsoft Notepad or Wordpad.

### To open a .C file with Browse

1. Choose **Simulate > Code Gen**.
2. In the Result File box, enter the name of the .C file to be opened.
3. Click on the Browse button.

# VisSim/C-Code Support Libraries

In addition to the program and utility files necessary to generate .C, .OBJ, .EXE, and .DLL files, VisSim/C-Code comes with two C support libraries:

CG32.LIB for the Windows platform

SIMOBJ.LIB for simulation object generation

During installation, VisSim/C-Code places the C support libraries in the \VISSIM80\CG and \VISSIM80\CG\LIB, respectively.

---

## Object files

Both C support libraries contain a collection of object files that contain compiled instructions to support blocks for which there is no direct translation into C source code. These blocks include:

- atan2
- buffer
- crossDetect
- dotProduct
- embed
- error
- export
- import
- integrator
- invert
- limitedIntegrator
- map
- multiply
- pulseTrain
- resetIntegrator
- stateSpace
- stop

- `timeDelay`
- `transferFunction`
- `transpose`
- `unitDelay`
- `vsum`

SIMOBJ.LIB also contains API instructions to support code generation.

# Targeting C Code for Unsupported Platforms

The source code for the C support library (CG32.LIB) is required for the following reasons:

- To enhance the functionality of the C support library.
- To generate executable files to be run on processors other than the ones supported by the object code version of the C support library shipped with VisSim/C-Code. For example, to embed the source code library in an Hitachi chip, you need to recompile and relink the support library using an Hitachi compiler.

The source code for the CG32.LIB support library is a separate product that is not automatically included when you purchase VisSim/C-Code.

---

## C support library source code

The source code for the C support library comprises the following files:

File name	Description
CG.C	Main driver routines
CG.H	Function prototypes
CGEN.H	Structure definitions
CGIO.C	File I/O
CGIO.H	Function prototypes
CROSSDET.C	Cross detection
CROSSDET.H	Function prototypes
File name	Description
FILEIO.C	File parsing
FILEIO.H	Function prototypes
IMPORT.C	File import/export
IMPORT.H	Function prototypes
MAT.C	Matrix operations
MAT.H	Function prototypes
MATDIV.C	Matrix divide

MATDIV.H	Function prototypes
READCC.TXT	ASCII text file containing additional technical information and corrections to the manual
SIMIO.H	Function prototypes
VCSRC.MAK	C code source makefile for Microsoft C v6.0
XFER.C	Transfer function support
XFER.H	Function prototypes
UNIX.MAK	Make file for Unix platforms
VCSRC.MAK	Project for Microsoft Visual C

---

## Copying the C support library source code to your hard disk

Copy the contents of the VisSim/C-Code Support Library Source Code disk to the code generation subdirectory.

---

## Compiling and linking the C support library source code

To compile and link the support library source code, you can use the makefile named SRC.MAK that was shipped with VisSim/C-Code. This makefile resides in the C:\VISSIM80\CG.

Platform	The makefile is configured to use
Windows	Microsoft Visual C 6.0+
UNIX	Gnu and native ANSI C compilers

**Note:** See CG32.LIB support library for more information.

### To compile and link the support library source code

- Enter one of the following commands at the system prompt:

To use this makefile	Use
Microsoft C	Open the project VCSRC.MAK
Gnu C or native ANSI C	make -f unixsrc.mak

# Installing VisSim/C-Code

This section contains.

---

## Installation requirements

VisSim/C-Code runs on personal computers using the Intel 80286 or higher processor, including the IBM Personal System/2 Series, the IBM PC AT, and 100% compatibles. To use VisSim/C-Code, your computer must have the following components:

- Visual Solutions VisSim 8.0+
- Microsoft Visual C6.x+ compiler
- 200K of free hard disk space

---

## Installing VisSim/C-Code

You use the Install program to install VisSim/C-Code on your computer for the first time or to upgrade to a more recent version of the software.

When you upgrade VisSim/C-Code, the installation program replaces old program and utility files with new ones. If there are existing files that you want to retain, Install gives you the opportunity to specify the files not to be overwritten.

### To install or upgrade VisSim/C-Code

1. Insert the VisSim/C-Code CD into the drive.
2. Do one of the following:
  - Click on **Start** and choose **Run**.
  - Select **File** from the **Program Manager** menu bar and choose the **Run** command.
  - In the Command Line box, type **A:\INSTALL** and click on the **OK** button, or press **ENTER**.
3. Follow the on-screen instructions.



# Index

## A

- Additional reading material 5
- Are you planning to port your C code to another platform 1
- Automatic DLL Generation 15

## B

- Basic preparation 7
- Building a custom DLL 19

## C

- C support library source code 33
- Calling a DLL from a VisSim diagram 17
- Checking for incomplete wiring 7
- Communicating with an embedded simulation object 24
- Comparing simulation speed 18
- Compiling and linking the C support library source code 34
- Copying the C support library source code to your hard disk 34
- Creating a DLL 15
- Creating a simulation object 23
- Creating a Stand-Alone Executable 11

## E

- Examining a .C file 30
- Exporting data 13

## G

- Generating an executable file 11
- Generating C code 29
- Generating code from automatically-generated DLLs 9
- Generating Source Code 29

## I

- Installation requirements 35
- Installing VisSim/C-Code 35
- Interactive webinars 4
- Introduction 1

## L

- LINK warning about LIBCLIB 21
- LINK warning messages that can be ignored 21

## O

- Object files 31
- Out of Environment Space error message 20
- Output data 12

## P

- Preparing for Code Generation 7
- Preparing for DLL or simulation object generation 8

## R

- Removing unsupported blocks 8
- Resetting the integration algorithm 8
- Resources for learning VisSim/C-Code 4
- Running an executable program 12

## S

- Sample diagrams 4
- Sample file with simulation object interface 26
- Simulation Object Generation 23

## T

- Targeting C Code for Unsupported Platforms 33
- The VisSim product family 2
- Training 4
- Translation of VisSim variable names 9
- Troubleshooting 20

## U

- Using the createSim function 25
- Using the vsmCgGetLastErrorString() 26
- Using the vsmCgRuntimeCommand 25

## V

- Verifying DLL results 18
- VisSim movies 4
- VisSim/C-Code Support Libraries 31
- VSM DLL32.BAT 9

## **W**

What is VisSim/C-Code 1

What you can do with DLLs 15

What you need in order to use VisSim/C-Code 1