# *VisSim/C-Code*

**Version 3**


**Visual Solutions**

INCORPORATED

**Visual Solutions, Inc.**

**VisSim/C-Code User's Guide - Version 3**



Visual Solutions, Inc.
487 Groton Road
Westford, MA O1886

# Contents


## Preface ..... v

Registering your software ..... v
Conventions used in this book ..... v
Getting help ..... vi
    Online help ..... vi
    Technical support service ..... vii
Additional reading material ..... vii

## Chapter 1 The Basics ..... 1

What you can do with 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 ..... 2

## Chapter 2 Preparing for Code Generation ..... 3

Basic preparation ..... 3
    Checking for incomplete wiring ..... 3
    Removing unsupported blocks ..... 4
    Resetting the integration algorithm ..... 5
Preparing for DLL generation ..... 5
Generating code from automatically-generated DLLs ..... 6
Generating code from illegal C identifier characters ..... 6

## Chapter 3 Creating a Stand-Alone Executable ..... 7

Generating an executable file ..... 7
Running an executable program ..... 9
    Output data ..... 9
    Exporting data ..... 9

## Chapter 4 Automatic DLL Generation .....11

What you can do with DLLs ..... 11
Creating a DLL ..... 12
Calling a DLL from a VisSim diagram ..... 14
Verifying DLL results ..... 15
Comparing simulation speed ..... 15
Building a custom DLL ..... 16

## Chapter 5 Generating Source Code.....19

Generating C code ..... 19
Examining a .C file ..... 21

## Appendix A Installing VisSim/C-Code.....23

Installation requirements ..... 23
Installation procedure ..... 24

## Appendix B VisSim/C-Code Support Library .....27

## Appendix C Targeting C Code for Unsupported Platforms .....29

C support library source code ..... 29
Copying the C support library source code to your hard disk ..... 30
Compiling and linking the C support library source code ..... 30

## Index .....33

# Preface

This manual describes how the components of VisSim/C-Code work together to provide you with a powerful environment for generating customizable ANSI C code from VisSim block diagrams and creating DLLs and executable files.

## Registering your software

Before you begin using VisSim, please fill out the enclosed registration card and mail it to us. As a registered user, you will receive a free subscription to The flexWire, along with discount promotions and VisSim workshop schedules.

## Conventions used in this book

This manual assumes that you are already familiar with the VisSim graphical user interface. If you need to review the interface, consult your *VisSim User's Guide*.

The following typographical conventions are used to make this manual:

| Visual convention | Where it's used |
|---|---|
| Shortcut key combinations | Shortcut key combinations are joined with a plus sign (+). For example, the command CTRL+C means hold down the CTRL key while you press the C key. |
| Hot keys | Hot keys are the underlined keys in VisSim's menus, commands, and dialog boxes. To use a hot key, press ALT and then the key for the underlined character. For instance, to execute the File menu's new command, hold down the ALT key while you press the F key, then release both keys and press the N key. |
| SMALL CAPS | To indicate the names of keys on the keyboard. |

| Visual convention | Where it's used |
|---|---|
| ALL CAPS | To indicate directory names, file names, and acronyms. |
| Initial Caps | To indicate menu names and command names. |
| `Lucida console` | To indicate block names. |

In addition, unless specifically stated otherwise, when you read "click the mouse..." or "click on...," it means to click the left mouse button.

# Getting help

To help you get the most out of VisSim, the following online information is available:

- **Online help** The online help contains step-by-step instructions for using VisSim features.
- **Online release notes** A file named READCC.TXT is installed in your main VisSim directory. This file contains last minute information and changes that were discovered after this manual went to print. For your convenience, you should read this file immediately and print a copy of it to keep with this manual.

## Online help

VisSim's Help program provides online instructions for using VisSim.

▶ **To open help**

- Do one of the following:

| To | Do this |
|---|---|
| Access the top level of help | Select Help from the menu bar or press ALT+H. |
| Access help on the selected block | Click on the Help button in the dialog box for the block. |

▶ **To close help**

- In the Help window, choose File > Exit, or press ALT+F4.

## Technical support service

When you need assistance with a Visual Solutions product, first look in the manual and read the online READCC.TXT file. If you cannot find the answer, contact the Technical Support group via a toll call between 9:00 am and 6:00 pm Eastern Standard Time, Monday through Friday, excluding holidays. The phone number is **978-392-0100.**

When you call in, please have the following information at hand:

- The version of VisSim and the version of the software operating environment that you're using
- The type of hardware that you're using
- All screen messages
- What you were doing when the problem happened
- How you tried to solve the problem

Visual Solutions also has the following fax and e-mail addresses:

| Address/Number | What it's for |
|---|---|
| 978-692-3102 | Fax number |
| bugs@vissol.com | Bug report |
| doc@vissol.com | Documentation errors and suggestions |
| sales@vissol.com | Sales, pricing, and general information |
| suggest@vissol.com | Product suggestions |
| tech@vissol.com | Technical support |

## 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).

# Chapter 1

# The Basics

This chapter covers the following information:

- What you can do with VisSim/C-Code
- How to port C code to another platform
- What you need to use VisSim/C-Code

## What you can do with 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

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

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. For more information on the VisSim/C-Code Support Library Source Code, see Appendix C, “Targeting C Code for Unsupported Platforms.”

## 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 C4.x or 5 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.

Chapter 2

# Preparing for Code Generation

This chapter covers the following information:

- Basic preparation to be performed on all block diagrams
- Additional preparation for generating DLLs
- Advanced preparation for generating code that includes DLLs

## 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 2nd

### 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.

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 are 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 | | ✓ |
| case | | ✓ |
| constraint | | ✓ |
| cost | | ✓ |
| DDE, DDEreceive, DDEsend | | ✓ |
| display | ✓ | |
| globalConstraint | | ✓ |
| histogram | | ✓ |
| light | ✓ | |
| lineDraw | | ✓ |

| Block name | Produce ASCII stream | Call EMPTY function |
|---|---|---|
| meter | ✓ | |
| neuralNet | | ✓ |
| parameterUnknown | | ✓ |
| plot | ✓ | |
| rt-DataIn* | ✓ | |
| rt-DataOut* | ✓ | |
| stop | ✓ | |
| 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 2nd order or Euler. VisSim/C-Code notifies you if it changes the integration algorithm.

# Preparing for DLL generation

If your goal is to create a DLL function, you must encapsulate the blocks to be converted into a DLL in a single compound block. If the compound block has numerous inputs and outputs, you should also label in the input and output connector tabs. Connector labels are carried over to the DLL making it easier to wire the DLL into the diagram.

▶ **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, automatically-generated DLLs in the portion of the diagram that is to be converted into an executable file, DLL function, or source code provided the DLL is declared in USERDLL.H and the library for the DLL is set 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 alias userlibs with the library file for the exported DLL function.

  To associate multiple libraries with userlibs, separate each library file with a comma. For example:

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

## Generating code from illegal C characters

The following characters are not legal C identifier characters: +, -, *, #, @, !. Although these characters are ASCII, they are not allowed in C variable names. During C code generation, these characters are converted into underscore (_) characters. This limitation is important when naming blocks that will eventually be compiled into C code.

Chapter 3

# Creating a Stand-Alone Executable

This chapter covers the following information:

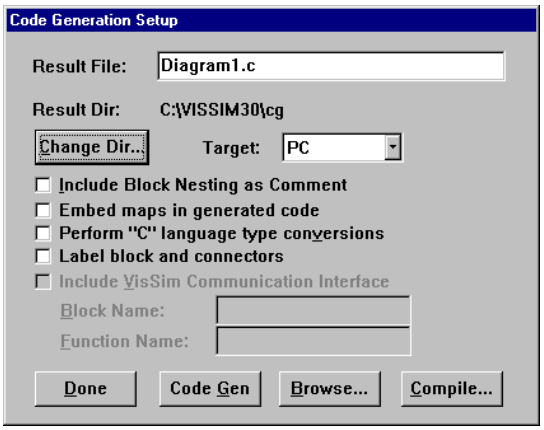
- Generating a stand-alone executable
- Running an executable file

## 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 as described in Chapter 2, “Preparing for Code Generation.”

▶ **To generate a stand-alone executable file**

1. Open the block diagram to be converted into an executable file.
2. Choose Simulate > Code Gen. The Code Generation Setup 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 Change Dir button and select a new directory.
5. The Target box contains the target platform for code generation. Choose the PC option, if it is not already selected.
6. Choose the following options you want. Options that do not apply are dimmed.

| 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 Generated 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. |
| Perform “C” Language Type Conversion | Use C language rules for determining result types. This has the effect of keeping the calculation in integer format longer and consequently results in more efficient run times on embedded processors |

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

When creating a stand-alone executable, the Label Block and Connectors, Include VisSim Communication Interface, Block Name, and Function Name 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 Setup 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 (`display`, `histogram`, `light`, `meter`, `plot`, or `stripChart`) 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*.

Chapter 4

# Automatic DLL Generation

This chapter covers the following information:

- Setting code options during compilation and linking
- Generating a DLL
- Altering the VisSim initialization file
- Calling a DLL from a VisSim diagram
- Verifying DLL results
- Comparing simulation speed
- Creating custom DLL dialog boxes

## 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.

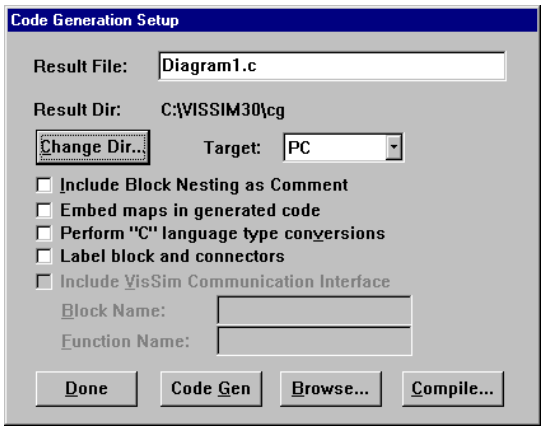
## Creating a DLL

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.

▶ **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, as described in Chapter 2, “Preparing for Code Generation.”
3. Select the compound block to be converted into a DLL.
4. Choose Simulate > Code Gen.

   The Code Generation Setup dialog box appears.




5. 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 Change Dir button.
7. The Target box contains the target platform for code generation. Choose the PC option, if it is not already selected.
8. Activate the Include VisSim Communication Interface option; then choose any of the other options you want.

| 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 Generated 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. |
| Perform “C” Language Type Conversion | Use C language rules for determining result types. This has the effect of keeping the calculation in integer format longer and consequently results in more efficient run times on embedded processors. |
| Label Block and Connectors | Include block and connector names with the DLL that indicate the source compound block in the VisSim diagram from which the DLL was created.<br><br>It is a good idea to activate this option, particularly when the resulting DLL has numerous input and output connector tabs. The connector names make it easy to identify the correct wiring paths. |
| Block Name | Specifies the name that will appear on the resulting userFunction block to which the DLL is bound. By default, the block name is the name of the compound block. |
| Function Name | Specifies the name of the DLL function. It defaults to cgMain. There is no need to edit this name. |

9. Click on the Compile button.
10. 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 Setup dialog box.
11. 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.

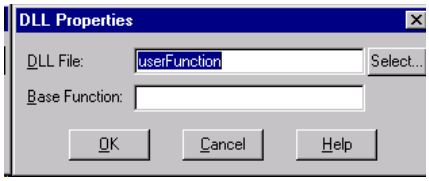
After you bind a DLL to a `userFunction` block, VisSim/C-Code renames the `userFunction` block with the DLL name. For example, if you created a DLL named AC Motor, VisSim/C-Code renames the `userFunction` block AC Motor.

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 Setup dialog box, as described on page 12. 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.

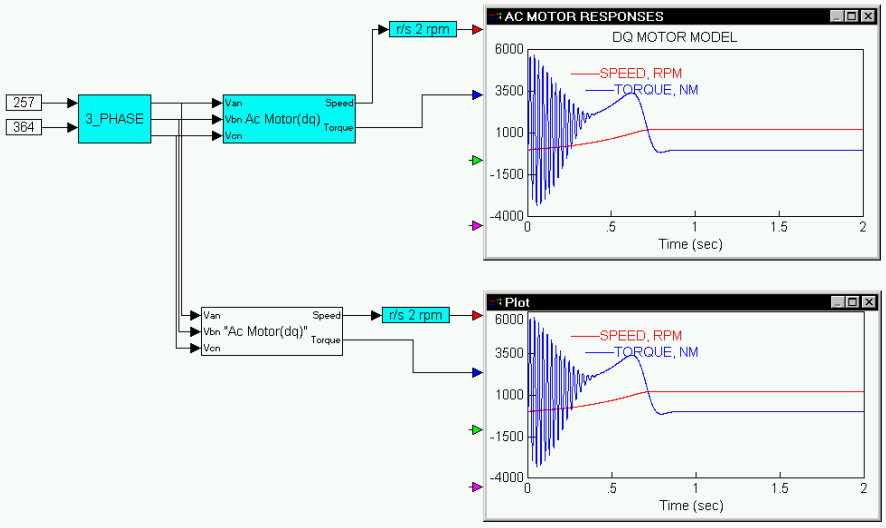
▶ **To view connector labels on a DLL**

- 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 procedure guides you through the process of building a project using the Microsoft Visual C++ v4 compiler. Refer to the documentation for the application language you are using for specific instructions.

▶ **To build a custom DLL with Microsoft Visual C++ v4**

1. Invoke the Compiler environment.
2. Do the following to create a new project workspace:
   - Choose File > New.
   - Under 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
- \VISSIM30\CG\LIB\CGDLL32.LIB
- \VISSIM30\VSDK\LIB\VISSIM32.LIB

4. Do the following to establish the settings for the build:
   - Choose Build > Settings.

     The Project Settings dialog box appears.
   - Click on the C/C++ tab.
   - Under Category, select Preprocessor.
   - In the Additional Include Directories box, specify the following:

     \VISSIM30\VSDK\INCLUDE,\VISSIM30\CG\INCLUDE
   - Click on the OK button.
5. Choose Build > Build to build the project.

### Troubleshooting

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++ v4 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.

Chapter 5

# Generating Source Code

This chapter covers the following information:

- Generating C code
- Editing existing C code
- Using a Project Build facility to create .EXE or .DLL files

## 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)

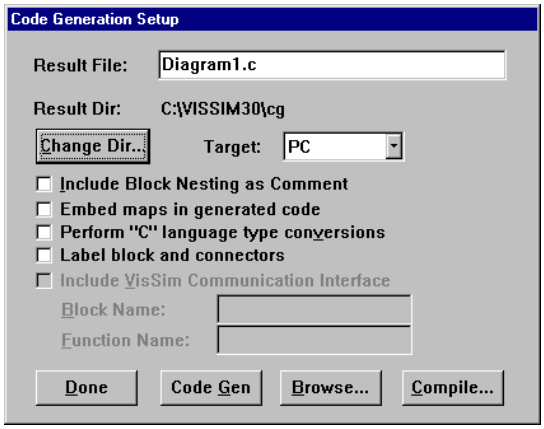
***Unsupported blocks***

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. These blocks are listed on page 4. 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 as described in Chapter 2, “Preparing for 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 PC 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 | "Generating an executable file" on page 7 for information on the options that you can select. |
| DLL | "Creating a DLL" on page 12 for information on the options that you can select. |

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.

Appendix A

# Installing VisSim/C-Code

The Install program that comes on your VisSim/C-Code disk installs the VisSim/C-Code program and other utility files on your hard disk. You use the Install program to:

- Install VisSim/C-Code on your computer for the first time
- Upgrade your existing copy of VisSim/C-Code to version 3

As part of the installation procedure, a READCC.TXT file is copied to the directory in which you install VisSim/C-Code. This file contains technical additions and corrections that were not available when this manual went to print. It's a good idea to read this file, and make a hard copy of it to keep with this user's guide.

## 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 3.0+
- Microsoft Visual C4.x or 5 compiler
- 200K of free hard disk space
- 3½" disk drive
- EGA or higher resolution monitor

## Installation procedure

You use the Install program to install VisSim/C-Code on your computer for the first time or to upgrade your existing copy of VisSim/C-Code 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

This procedure assumes that you are installing from drive A to your hard disk. If you are installing from a different drive, substitute the correct drive designation in the installation procedure.

1. Start Windows.
2. Insert the disk labeled VisSim/C-Code into drive A.
3. Do one of the following:
   - Click on Start and choose Run.
   - Select File from the Program Manager menu bar and choose the Run command.
4. In the Command Line box, type A:INSTALL and click on the OK button, or press ENTER.
5. An Install dialog box appears.

   Install asks you where you want to install VisSim/C-Code. You can accept the default path or type in a different directory. Make sure that the VisSim/C-Code files are installed on the same disk and directory that contain your VISSIM.EXE.

   If you are installing VisSim/C-Code over an earlier version, the Install program will replace the old utility files with new ones, and retain all user-written files. If you've made changes to old files that were supplied with VisSim/C-Code, you can request that Install ask for confirmation before it overwrites each file. An X in the Ask Before OverWriting Existing Files check box activates this option.
6. To accept the information in the dialog box, click on the Continue button, or press ENTER.

   Install installs an updated VisSim executable in the directory you specified. It also creates a subdirectory named \CG in which it installs code-generation-related files.

When the installation is complete, VisSim displays a dialog box indicating that VisSim/C-Code has been successfully installed.

7. Click on the OK button, or press ENTER.

## Appendix B

# VisSim/C-Code Support Library

In addition to the program and utility files necessary to generate .C, .OBJ, .EXE, and .DLL files, VisSim/C-Code comes with a C support library (CG32.LIB) for the Windows platform. During installation, VisSim/C-Code places the C support library in the \VISSIM30\CG directory.

The C support library is 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

Appendix C

# Targeting C Code for Unsupported Platforms

The source code for the C support library 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 C 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:\VISSIM30\CG.

| Platform | The makefile is configured to use |
|---|---|
| Windows | Microsoft Visual C 4.0+ |
| UNIX | Gnu and native ANSI C compilers |

**Note:** For information about the C support library, refer to Appendix B, “VisSim/C-Code Support Library.”

▶ **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 |

# Index


## B

Block Name option, 8, 13
browsing C code, 21

## C

C code
  browsing, 21
C code generation
  illegal characters, 6
  integration algorithm, 5
  porting, 1
  procedure, 19
  unsupported blocks, 4
  uses, 1
  wiring, 3
C compiler, choosing, 2
C identifer characters, illegal, 6
C support library, 27
C support library source code, 29–31
  compiling and linking, 30
  copying to disk, 30
CG32.LIB. *See* C support library
checking wiring connections, 3
Code Gen (Simulate), 7, 12, 19
compound blocks, for DLL generation, 5
Connector Labels (View), 15
conventions, v

## D

DLLs
  binding to userFunction block, 14
  building custom dialog boxes, 16
  calling from VisSim, 14
  comparing performance, 15
  compound blocks, 5
  creating, 12
DLLs (*continued*)
  generating code from, 6
  uses, 11
  verification, 15

## E

Embed Maps in Generated Code option, 8, 13

## F

Function Name option, 8, 13

## G

generating C code, 19
  customizing, 21
  from automatically-generated DLLs, 6
  illegal characters, 6
  integration algorithm, 5
  unsupported blocks, 4
  wiring, 3
generating stand-alone exectuables, 7

## I

Include Block Nesting as Comment option, 8, 13
Include VisSim Communication Interface option, 8, 13
installing VisSim/C-Code
  procedure, 24
  requirements, 23

## L

Label Block and Connectors option, 8, 13

## M

multi-rate execution, 11

## O

online help, vi

## P

Perform C Language Type Conversion option, 8, 13
porting C code, 1
preparing for code generation
  integration algorithm, 5
  unsupported blocks, 4
  wiring, 3
preparing for DLL generation
  creating compound blocks, 5
Project Build facilities, 16
protecting intellectual property, 12

## R

READCC.TXT, vi, 23
registration, v

## S

speeding up simulation time, 11
stand-alone executables
  exporting data, 9
  generating, 7
  output data, 9
  running, 9

## T

technical support, vii

## U

unsupported blocks, 4
userFunction block
  setting up, 14