

Getting Started with the NI LabVIEW™ C Generator

Use the LabVIEW C Generator to generate generic C code from VIs. The generated C code can run on most platforms.



Note Refer to the *LabVIEW C Generator Readme*, which is available by opening `readme_C_Generator.html` on the NI LabVIEW C Generator CD, for system requirements and instructions about installing the C Generator.

This manual contains a step-by-step tutorial that shows you how to generate C code from an exported VI and use that generated C code on a target. In this tutorial, you specify a VI to export for C code generation, configure C code generation settings, define the function prototype for the exported VI, generate C code, and use the generated C code on a non-LabVIEW target. The tutorial also describes how to create and configure the generated C code so you can use it in the following commonly used targets:

- A 32-bit Windows platform (Win32) DLL created with the Microsoft Visual Studio compiler.
- A static library created in Cygwin using GNU Compiler Collection (GCC).
- An ARM application built in Keil μ Vision for a Luminary Micro EK-LM3S8962 target.



Note You must configure some settings differently for each target. As you use this tutorial, refer to the *Win32 DLL*, *Cygwin Static Library*, and *EK-LM3S8962 Application* sections of this manual for target-specific instructions.

Contents

Installing Additional Third-Party Software.....	1
Creating the LabVIEW Project.....	2
Creating a Build Specification.....	2
Creating a VI for C Code Generation.....	3
Defining the Function Prototype.....	5
Setting C Code Generation Options.....	7
Testing the VI.....	7
Generating C Code in LabVIEW.....	9
Building the Generated C Code.....	11
Win32 DLL.....	11
Cygwin Static Library.....	13
EK-LM3S8962 Application.....	15
Where to Go from Here.....	16

Installing Additional Third-Party Software

You must install the following third-party software based on the target on which you want to use the generated C code:

- **Win32 DLL**
 - Microsoft Visual Studio 2008 compiler—Refer to the Microsoft Web site at www.microsoft.com for more information about installing Microsoft Visual Studio.

- GNU Make for Windows—Refer to the Make for Windows Web site at <http://gnuwin32.sourceforge.net/packages/make.htm> for more information about installing the GNU Make utility.
- **Cygwin Static Library**
 - Cygwin 1.7.5 or later—Refer to the Cygwin Web site at www.cygwin.com for more information about installing Cygwin. Install Cygwin with the following packages:
 - binutils 2.20.51 or later
 - gcc-g++ 3.4.4 or later
 - make 3.81 or later
- **EK-LM3S8962 Application**
 - RealView Microcontroller Development Kit, including Keil μ Vision3 or later—Refer to the Keil Web site at www.keil.com for more information about installing Keil μ Vision.

Creating the LabVIEW Project

Use LabVIEW projects (.lvproj) to group together LabVIEW files and non-LabVIEW files, create build specifications for exporting VIs for C code generation, and generate C code. You must use a LabVIEW project to generate C code.

Complete the following steps to create a project.

1. Launch LabVIEW.
2. Select **File»New Project** to display the **Project Explorer** window.

The **Project Explorer** window displays a new untitled project, which includes the **My Computer** target. The **Project Explorer** window includes two pages, the **Items** page and the **Files** page. The **Items** page displays the project items as they exist in the project tree. The **Files** page displays the project items that have a corresponding file on disk. Project items on the **Files** page both reflect and update the contents on disk. You can switch from one page to the other by clicking the **Items** and **Files** tabs.

3. Right-click the untitled project in the **Project Explorer** window and select **Save** from the shortcut menu.
4. In the **Name the Project** dialog box, enter `GreatestCommonDivisor` and select a location to save the project file.
5. Click the **OK** button to save the project.

Creating a Build Specification

You use a build specification to specify how the C Generator generates C code for exported VIs. You can have multiple build specifications under the same **My Computer** target. For example, if you want to generate C code from an exported VI for a Win32 DLL and a Luminary Micro EK-LM3S8962 target, you can create a separate build specification for each of these targets.

Complete the following steps to create the build specification.

1. In the **Project Explorer** window, right-click **Build Specifications** under **My Computer** and select **New»C Code Generation** from the shortcut menu to create a C code generation build specification. LabVIEW displays the **C Code Generation Properties** dialog box.
2. On the **Information** page of the **C Code Generation Properties** dialog box, enter `Greatest Common Divisor` in the **Build specification name** text box to rename the build specification.
3. In the **Destination directory** text box, specify a location to save the generated C files.

(EK-LM3S8962 Application) Set the value of the **Destination directory** option to `labview\examples\CGenerator\Tutorial\Keil - ARM LM3S8962`. The C Generator installs an example μ Vision project (.UV2) you will use in this tutorial to build the generated C code into an ARM application. The μ Vision project includes the generated C code from this location.

4. Verify that the **Information** page appears similar to the following figure.

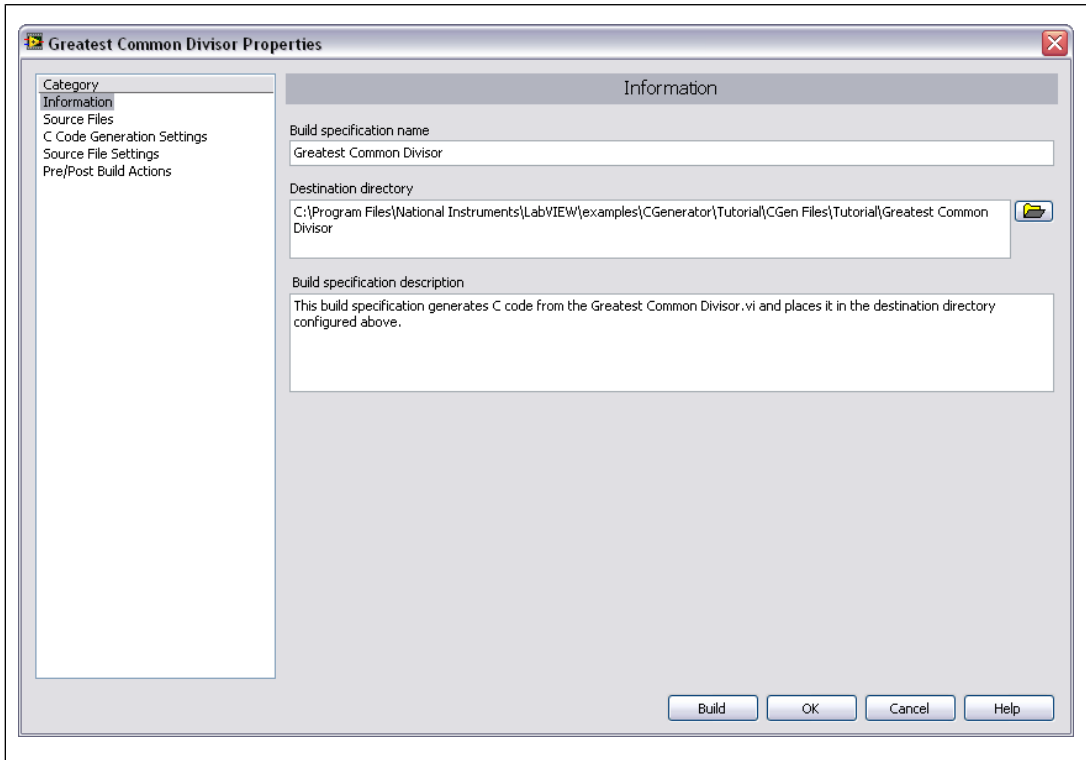


Figure 1. Defining the Build Specification Information

5. Click the **OK** button to close the build specification.
6. LabVIEW displays a dialog box to notify you that you have not selected a VI to export. Click the **Yes** button.
7. Save the project.

Creating a VI for C Code Generation

After you create a LabVIEW project and C code generation build specification, you can create a new VI or add an existing VI to the project. This tutorial uses an existing example VI, `Greatest Common Divisor.vi`, which is installed with the C Generator.

Complete the following steps to add the VI to the project.

1. In the **Project Explorer** window, right-click **My Computer** and select **Add»File** from the shortcut menu.



Note You must include the VI you want to generate C code for under **My Computer**. You cannot generate C code from any other targets in the **Project Explorer** window.

2. In the **Select a File to Insert** dialog box, navigate to `labview\examples\CGenerator\Tutorial\Greatest Common Divisor.vi`. Click the **Add File** button. The VI appears under **My Computer**, as shown in the following figure.

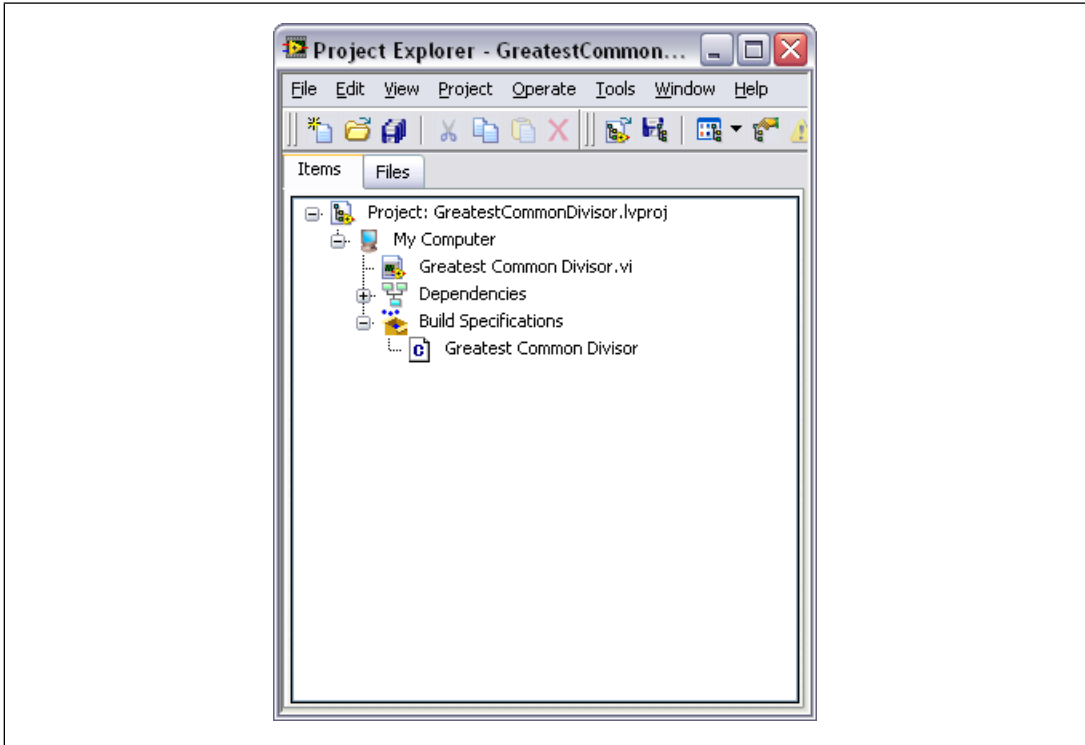


Figure 2. Adding the VI to the Project

3. Save the project.
4. Open the VI and view the block diagram, which is shown in the following figure.

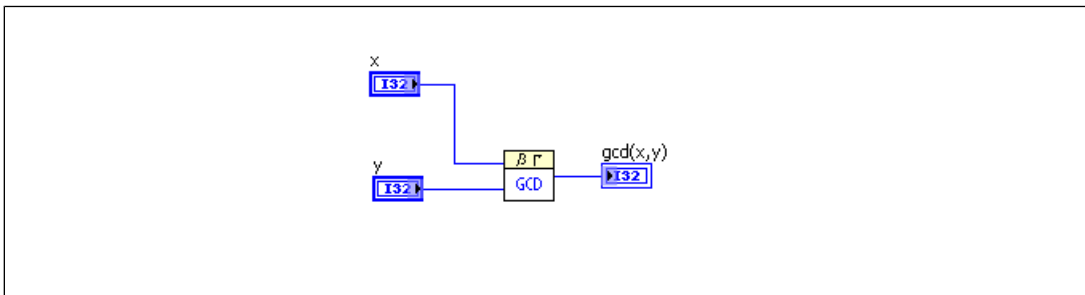


Figure 3. Viewing the Greatest Common Divisor.vi Block Diagram

The `Greatest Common Divisor.vi` computes the greatest common divisor between two numbers.



Tip When you create a new VI to export for C code generation, you must account for differences in VIs you develop to run as LabVIEW applications and VIs you develop for C code generation. Exported VIs and all VIs in their hierarchies support some block diagram objects differently from VIs running as LabVIEW applications. Refer to *Developing VIs for C Code Generation* in the *LabVIEW Help* for more information about these differences. Select **Help»LabVIEW Help** to search the *LabVIEW Help*.

Defining the Function Prototype

Use the build specification to specify which VI you want to export for C code generation and to define the function prototype that the C Generator generates for that VI.

Complete the following steps to define the function prototype.

1. Double-click the **Greatest Common Divisor** build specification in the **Project Explorer** window to display the **C Code Generation Properties** dialog box.
2. Select **Source Files** from the **Category** list in the **C Code Generation Properties** dialog box to display the **Source Files** page.
3. From the **Project Files** tree, select **GreatestCommonDivisor.vi** and click the **Add Item** arrow button next to the **Exported VI** listbox to specify the VI you want to export for C code generation, as shown in the following figure.

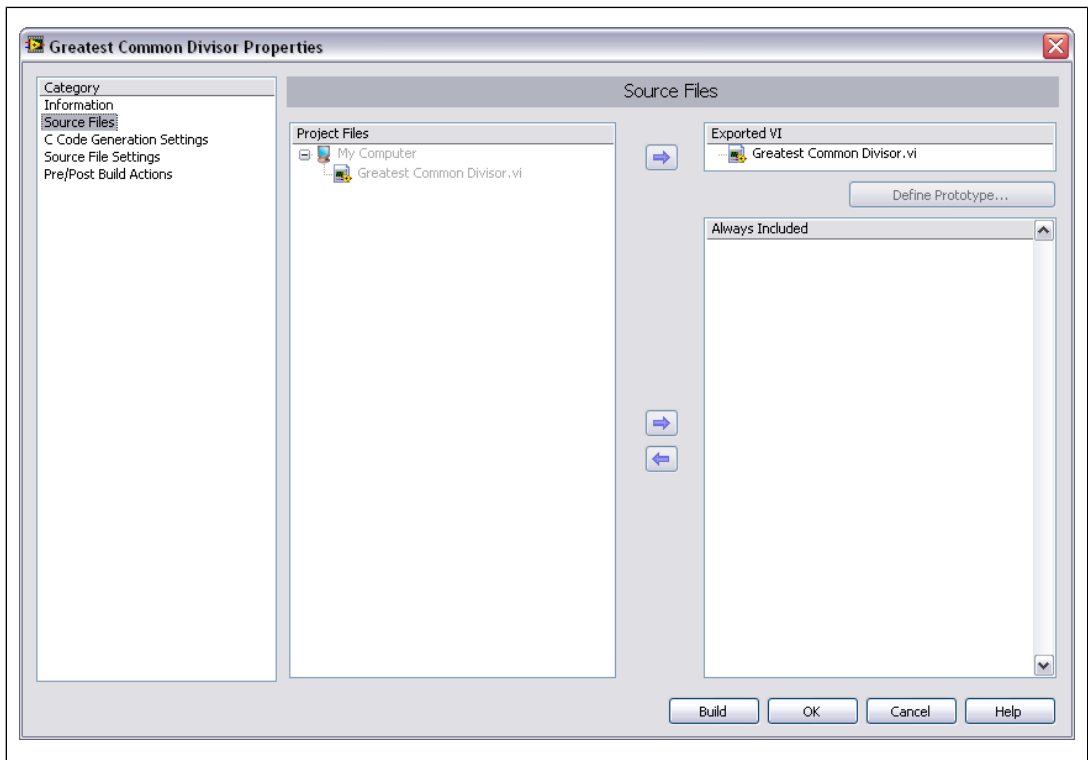


Figure 4. Specifying a VI to Export for C Code Generation

LabVIEW automatically displays the **Define VI Prototype** dialog box when you specify a VI in the **Exported VI** listbox. Use the **Define VI Prototype** dialog box to define the parameters of the function prototype that the C Generator creates from the exported VI. Notice that the text displayed in the **Function Name** text box—`GreatestCommonDivisor`—matches the name of the VI you selected for export. Also notice that the parameters displayed in the **Parameters** list match the inputs and outputs of the exported VI. The **Function Prototype** text box displays a preview of the function prototype that the C Generator generates based on the current settings in the dialog box.

4. Select **returnvalue** in the **Parameters** list of the **Define VI Prototype** dialog box.
5. In the **Current Parameter** section, set the **VI Output** option to **gcdxy**. The **gcdxy** output of the `Greatest Common Divisor.vi` is now the return value of the function prototype. The **Function Prototype** text box in the **Define VI Prototype** dialog box, displays a preview of the function prototype, as shown in the following figure.

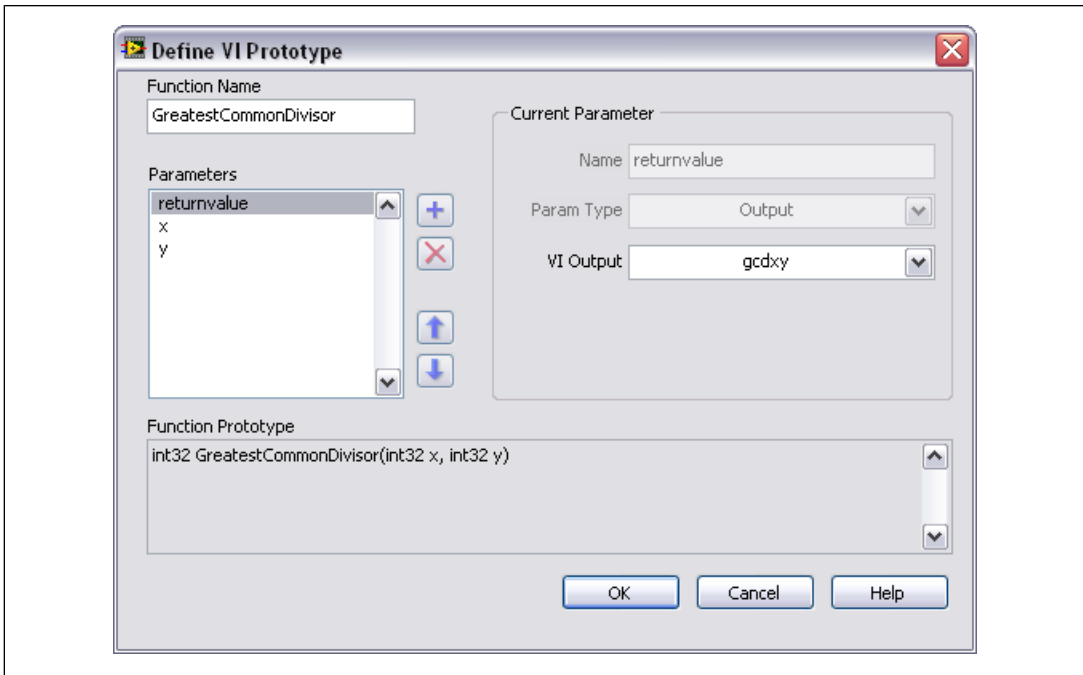


Figure 5. Defining the Function Prototype

6. Click the **OK** button to close the **Define VI Prototype** dialog box.



Note When you create a VI you want to export for C code generation, you must select a pattern to use for the connector pane and assign all front panel controls and indicators to the connector pane terminals. If you do not set up the connector pane for the exported VI, you cannot configure parameters for the function prototype in the **Define VI Prototype** dialog box. Refer to *Assigning Terminals to Controls and Indicators* in the *LabVIEW Help* for more information about setting up the connector pane for a VI.

Setting C Code Generation Options

From a LabVIEW block diagram, the C Generator creates ANSI C code that you can compile on many targets. You must specify platform-specific settings for the target on which you want to use the generated C code.

Complete the following steps to set C code generation options for the target.

1. Select **C Code Generation Settings** from the **Category** list in the **C Code Generation Properties** dialog box to display the **C Code Generation Settings** page.
2. Specify the endianness and alignment for the target on which you want to use the generated C code.

(Win32 DLL) Set the **Endian format** option to **Little endian** and the **Alignment** option to **4**.

(Cygwin Static Library) Set the **Endian format** option to **Little endian** and the **Alignment** option to **8**.

(EK-LM3S8962 Application) Set the **Endian format** option to **Big endian** and the **Alignment** option to **8**.



Note Refer to *Determining Endianness and Alignment* in the *LabVIEW Help* for more information about determining endianness and alignment for the target on which you want to use generated C code.



3. Click the **OK** button to close the dialog box.

Testing the VI

National Instruments recommends you test the exported VI before generating C code and test generated C code before you use it externally. You can test the exported VI in LabVIEW to simulate the behavior of the generated C code.

The C Generator includes syntax checking of the exported VI. When you create a C Code Generation build specification, the C Generator enables syntax checking for the exported VI. When LabVIEW checks the syntax of the VI for which you want to generate C code, LabVIEW is checking that the C Generator can create generic C code based on the design of the exported VI. LabVIEW always checks the syntax of the exported VI before it builds the VI.

Complete the following steps to test the exported VI in LabVIEW and check the syntax of the exported VI.

1. Open `Greatest Common Divisor.vi`.
2. Enter two numbers in the **x** and **y** controls. Click the **Run** button, , in the VI. The VI displays the greatest common divisor of the two numbers.
3. View the block diagram. If the block diagram is not visible, select **Window»Show Block Diagram**.
4. Add a Timed Loop to the block diagram. The Timed Loop is located on the **Timed Structures** palette. Because the Timed Loop is unsupported by the C Generator, LabVIEW displays the **Warning** button, , in the VI toolbar, as shown in the following figure.

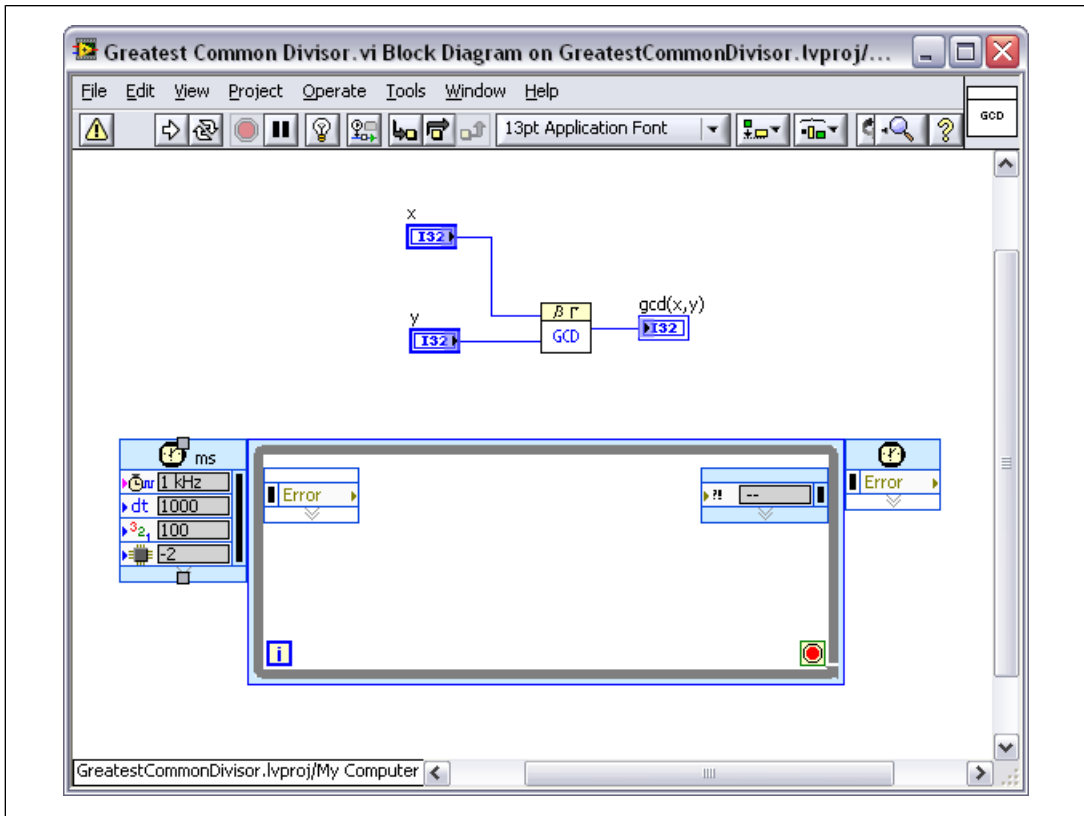


Figure 6. Displaying Warnings on the Block Diagram

LabVIEW displays the **Warning** button during syntax checking if the exported VI hierarchy contains unsupported data types, VIs, functions, or structures. While syntax warnings do not prevent you from running the exported VI in LabVIEW, you cannot generate C code for the VI until you correct all syntax warnings.

5. Click the **Warning** button in the VI toolbar to display the **Error list** window, which explains that the Timed Loop is unsupported by the C Generator, as shown in the following figure.

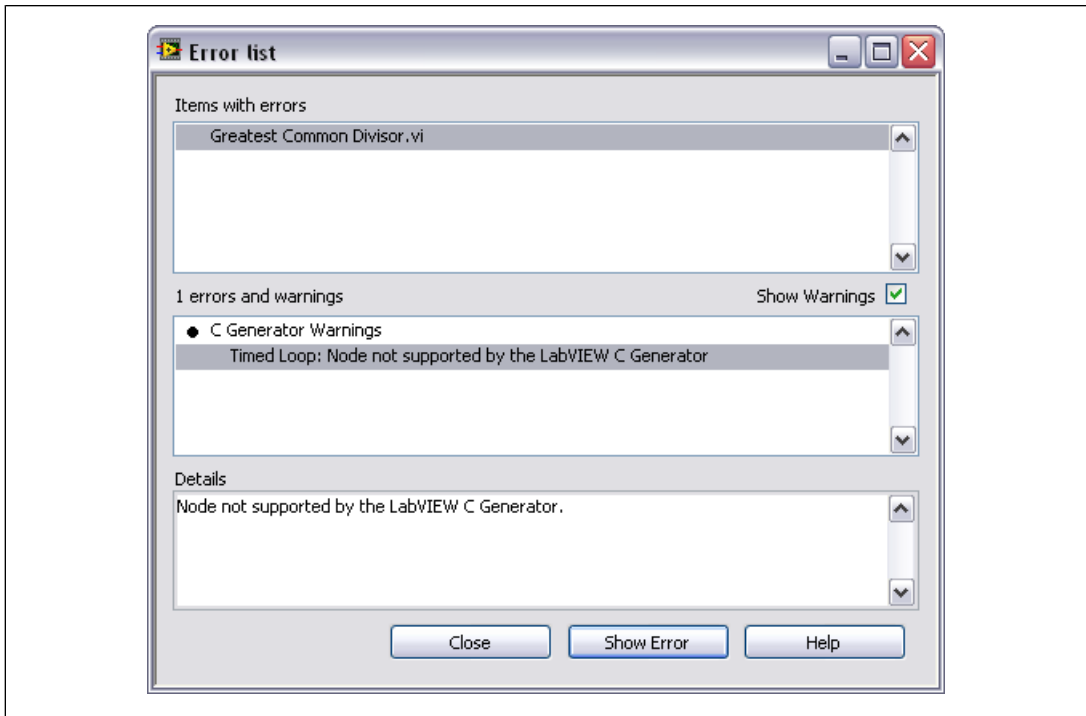


Figure 7. Displaying Warnings in the Error List Window

- Remove the Timed Loop from the block diagram. LabVIEW no longer displays the **Warning** button in the VI toolbar because the syntax is correct.



Note If you remove the checkmark from the **Show Warnings** checkbox in the **Error list** window, LabVIEW no longer displays syntax warnings for C code generation. If you disable warnings, you can check the syntax of the exported VI by right-clicking the C Code Generation build specification in the **Project Explorer** window and selecting **Check Syntax** from the shortcut menu.

Generating C Code in LabVIEW

The LabVIEW C Generator uses the options you specify in the **C Code Generation Properties** dialog box to generate C code from the block diagram. The C Generator generates each VI in the VI hierarchy into a separate C file using a C function name, which is more restrictive than a VI name. Any non-alphanumeric characters become underscores. If the VI name begins with a non-alphanumeric character, the C Generator prepends `A_` to the beginning of the C function name.

Complete the following steps to generate C code from the exported VI.

- Expand **Build Specifications** in the **Project Explorer** window.
- Right-click the **Greatest Common Divisor** build specification and select **Build** from the shortcut menu.



Tip You can check the syntax of the exported VI before you build by right-clicking the build specification and selecting **Check Syntax** from the shortcut menu.

LabVIEW displays the **Build status** dialog box while it generates C code for the exported VI.

3. Click the **Explore** button in the **Build status** dialog box to open the directory containing the generated C code. The C Generator generates the C code to the location you specified in the **Destination directory** option on the **Information** page of the **C Code Generation Properties** dialog box.



Tip To open the directory containing the generated C files, you also can right-click the C Code Generation build specification in the **Project Explorer** window and select **Explore** from the shortcut menu.

The C Generator generates the following files:

- `Greatest_Common_Divisor.c` contains the exported function. You must include this file in the linker input list.
- `Greatest_Common_DivisorLib.h` contains the library interface that specifies the generated function you defined in the **Define VI Prototype** dialog box. You must include this file in the external application.
- `NI_GMath_lvlib_Gcd.c` contains an internal function that implements a subVI, `Gcd.vi`. You must include this file in the linker input list.
- `LVForms.h` contains declarations for the entry point functions for each VI. This file is required for compiling the generated C files.
- `LVFuncsUsed.h` contains usage information to support source-level dead-code stripping. This file is required for compiling the generated C files.
- `LVGlobs.h` contains declarations for global initialization functions and data type information used in an external application. This file is required for compiling the generated C files.
- `Makefile` is an example makefile. When you use the `make` command, the GNU Make utility reads this file. After the C Generator generates `Makefile`, the C Generator does not overwrite this file if you regenerate the C code. If you want to generate a new example `Makefile`, you must delete the file before regenerating the C code.



Note LabVIEW code depends on the LabVIEW run-time library, which includes support for all basic functions. You must link to the LabVIEW run-time library with all exported VIs. The LabVIEW run-time library source is located in the `labview\CCodeGen\libsrc` directory. You also must link to the LabVIEW analysis library if the exported VI uses math or signal processing VIs. The LabVIEW analysis library source is located in the `labview\CCodeGen\analysis` directory. You also must link to the LabVIEW simulation library if the exported VI uses the Simulation VIs and functions. The LabVIEW simulation library source is located in the `labview\CCodeGen\simulation`.

- `dependencies.mk` is the secondary makefile. The C Generator includes all LabVIEW settings, such as the build specification name and C code generation option values, that are required to generate C code. The C Generator updates this file if the hierarchy of the exported VI changes. LabVIEW also automatically overwrites this file during the build process.

`DllMain.c`, which is not created when the C Generator generates C code, is an example entry point for a DLL. This file is required for building DLLs that run on Windows. This file is located in the `labview\CCodeGen\libsrc\platform\win` directory.

Building the Generated C Code

In general, you include the generated C code on a target in one of the following ways:

- Add the necessary files directly into the C project.
- Build the generated C code files into a static library, or archive file, and link the static library into an application.
- Build the generated C code files into a DLL and load the DLL from an application at run time.

Before you use the C code that the C Generator generates from the exported VI, you must configure the target. Refer to the *Win32 DLL*, *Cygwin Static Library*, and *EK-LM3S8962 Application* sections of this manual for target-specific instructions on including the generated C code.



Note You must install third-party software to complete this part of the tutorial. Refer to the *Installing Additional Third-Party Software* section of this manual for more information about additional software.

Win32 DLL

Complete the following steps to use the generated C code in a Win32 DLL created with the Microsoft Visual Studio compiler.



Note By default, the C Generator creates a `Makefile` for a Win32 DLL created with the Microsoft Visual Studio compiler.

1. Open `Makefile`, which is located in the directory that contains the generated C files.
2. Ensure that the following source files are included:

```
BDLIBS1 = CCGArrSupport.c CCGArrSupport2.c CCGArrSupport3.c
CCGClusterSupport.c CCGCmplxSupport.c CCGEnumSupport.c CCGDVRSupport.c
CCGFXPSupport.c

BDLIBS2 = CCGFltSupport.c CCGIntSupport.c CCGStrSupport.c CCGTimeSupport.c
CCGTokString.c CCGUDClassSupport.c CCGVariant.c CCGXMLSupport.c ExecStack.c

BDLIBS3 = LVBlockDiagram.c LVCGenRTInit.c LVContext.c LVHeap.c LVTdp.c
MemCheck.c nbemptybin.c nbfifo.c nbfifo_inst.c nbitemtable.c NumText.c
ViLib.c rtmath.c

BDLIBS = $(BDLIBS1) $(BDLIBS2) $(BDLIBS3)

FPLIBS1 = LVArrayControl.c LVBoolean.c LVClusterControl.c LVEnumCtl.c
LVFrontPanel.c

FPLIBS2 = LVListbox.c LVNumeric.c LVRadioClustCtl.c LVRing.c LVScrollbar.c
LVString.c LVTab.c LVTable.c

FPLIBS3 = LVTimestamp.c LVTree.c Pict.c

FPLIBS = $(FPLIBS1) $(FPLIBS2) $(FPLIBS3)

COMMSLIBS = CCGVIRefSupport.c crc.c

LVLIBS = arrresize.c
```

COMMONLIBS = Headless.c



Tip Refer to *Including Generated C Code in External Applications* in the *LabVIEW Help* for more information about which source files you must compile.

3. Ensure that the following compiler flags are set:

```
CLFLAGS = /D "CGEN_LIBFUNC_PREFIX=__declspec(dllexport)" /D  
"_CRT_SECURE_NO_WARNINGS" /MT /Zp4 /W3 /nologo /c
```

The `/c` flag is set to compile C code. The `/Zp4` flag sets the alignment to 4 for allocating structs. The `/D "CGEN_LIBFUNC_PREFIX=__declspec(dllexport)" /D` preprocessor define also must be set for the LabVIEW run-time library. You do not need any OS- or CPU-specific preprocessor defines for the LabVIEW run-time library.

4. Ensure the following include paths are set:

```
INCLUDES = /I "." /I "$(LVPATH)\CCodeGen\include\blockdiagram" /I  
"$(LVPATH)\CCodeGen\include\comms" /I  
"$(LVPATH)\CCodeGen\include\frontpanel" /I  
"$(LVPATH)\CCodeGen\include\platform\win" /I  
"$(LVPATH)\CCodeGen\analysis\development\include" /I  
"$(LVPATH)\CCodeGen\analysis\LV\source\include"
```

5. Save and close the Makefile.
6. Open the LabVIEW analysis library Makefile, which is located in the `labview\CCodeGen\analysis` directory.
7. The LabVIEW analysis library Makefile must include OS- and CPU-specific preprocessor defines. The following preprocessor defines are set:

```
CLFLAGS = /D "LV_Embedded" /D "NoFFTtablePersist=1" /D  
"COMPILE_FOR_SMALL_RT=1"
```

8. Close the Makefile.
9. Open and review `LVDefs_plat.h`, which is located in the `labview\CCodeGen\include\platform\win` directory.

`LVDefs_plat.h` contains basic constants and macros that the C Generator needs to generate C code. This file provides the mapping between generic C function calls that the C Generator generates and platform-specific, run-time functions. `LVDefs_plat.h` also contains platform definitions of data types for the compiler.

To properly define data types, you must know the compiler definitions of those data types. To define the generic function calls, you must know how the target SDK implements these function calls. `LVDefs_plat.h` defines the data types and function calls correctly for a Win32 DLL created with the Microsoft Visual Studio compiler.

10. Close `LVDefs_plat.h`.
11. Open and review `LVSysIncludes.h`, which is located in the `labview\CCodeGen\include\platform\win` directory.

`LVSysIncludes.h` contains platform-specific header files, toolchain-specific header files, and standard C header files. Most files in the `labview\CCodeGen` directory include `LVSysIncludes.h`.

12. Close `LVSysIncludes.h`.

Running the GNU Make Utility to Create a Win32 DLL

You use the GNU Make for Windows utility to build applications and libraries automatically from the generated C code. Before you can run the `Makefile` in the directory that contains the generated C code, you must set up the required toolchain.

Complete the following steps to set up the toolchain and run the GNU Make utility.

1. Set up the `PATH` environment variable in Windows to include `make.exe`.
2. Ensure the `VSPATH` variable in `Makefile` specifies the location of the Microsoft Visual Studio compiler, `cl.exe`.
3. Run the Visual Studio Command Prompt to set up the environment for the compiler. Refer to the Microsoft Visual Studio compiler documentation for more information about setting up the environment.
4. Call the `make` command from the command line to run the GNU Make utility on the `Makefile` in the directory that contains the generated C code.

Running and Calling the Generated C Code

Now you can call the Win32 DLL. You can declare the exported function prototype as shown in the following example:

```
extern int GreatestCommonDivisor(int x, int y);
```

You can call the exported function as shown in the following example:

```
gcd = GreatestCommonDivisor(12,15);
```

To load the DLL automatically when the application starts, you can add `Sharedlib.lib` to the linker inputs. The GNU Make utility generates `Sharedlib.lib`.

Cygwin Static Library

Complete the following steps to use the generated C code in a static library created in Cygwin using GCC.

1. Open the directory that contains the generated C code files.
2. Open the `labview\examples\CGenerator\Tutorial\Cygwin` directory.
3. Replace the `Makefile` in the directory that contains the generated C code with the `labview\examples\CGenerator\Tutorial\Cygwin\Makefile`.

When the C Generator generates C code, the C Generator always creates a `Makefile` for a Win32 DLL created with the Microsoft Visual Studio compiler and a `Makefile.cygwin` for a static library created with Cygwin. You must replace the generated makefiles with the example `Makefile`, which is customized for this tutorial.

4. Copy `labview\examples\CGenerator\Tutorial\Cygwin\main.c` into the directory that contains the generated C code.
5. Open the `Makefile`.
6. Ensure that the following source files are included:

```
BDLIBS1 = CCGArrSupport.c CCGArrSupport2.c CCGArrSupport3.c  
CCGClusterSupport.c CCGCmplxSupport.c CCGEnumSupport.c CCGDVRSupport.c  
CCGFXPSupport.c
```

```
BDLIBS2 = CCGFltSupport.c CCGIntSupport.c CCGStrSupport.c CCGTimeSupport.c  
CCGTokString.c CCGUDClassSupport.c CCGVariant.c CCGXMLSupport.c ExecStack.c
```

```
BDLIBS3 = LVBlockDiagram.c LVGenRTInit.c LVContext.c LVHeap.c LVTdp.c
MemCheck.c NumText.c ViLib.c rtmath.c
```

```
BDLIBS = $(BDLIBS1) $(BDLIBS2) $(BDLIBS3)
```

```
FPLIBS1 = LVArrayControl.c LVBoolean.c LVClusterControl.c LVEnumCtl.c
LVFrontPanel.c
```

```
FPLIBS2 = LVListBox.c LVNumeric.c LVRadioClustCtl.c LVRing.c LVScrollbar.c
LVString.c LVTab.c LVTable.c
```

```
FPLIBS3 = LVTimestamp.c LVTree.c Pict.c
```

```
FPLIBS = $(FPLIBS1) $(FPLIBS2) $(FPLIBS3)
```

```
COMMSLIBS = CCGVIRefSupport.c crc.c
```

```
LVLIBS = arrresize.c
```

```
COMMONLIBS = Headless.c
```



Tip Refer to *Including Generated C Code in External Applications* in the *LabVIEW Help* for more information about which source files you must compile.

7. Ensure that the following compiler flag is set:

```
CFLAGS = -O2
```

The `-O2` flag is set to optimize the C code and is an optional flag.



Tip If you are using an x86 target, you must set the `-malign-double` GCC command line option. You also must set the **Alignment** option to 8 on the **C Code Generation Settings** page of the **C Code Generation Properties** dialog box.

8. Ensure the following include paths are set:

```
INCLUDES = -I$(LVPATH)/CCodeGen/include/platform/cygwin-gcc
-I$(LVPATH)/CCodeGen/include/comms
-I$(LVPATH)/CCodeGen/include/blockdiagram \
-I$(LVPATH)/CCodeGen/include/frontpanel -I.
```

9. Notice that the `Makefile` also includes the following settings:

```
.DEFAULT_GOAL = default
```

```
default: $(APPLICATION).a
```

```
GCD.exe: main.o $(APPLICATION).a
```

```
$(LD) $(LDFLAGS) -lm -o $@ main.o $(APPLICATION).a
```

By default, running the `Makefile` creates a `.a` file that you can link to your application. The `Makefile` also can create an executable program, `GCD.exe`.

10. Save and close the `Makefile`.
11. Open and review `LVDefs_plat.h`, which is located in the `labview\CCodeGen\include\platform\cygwin-gcc` directory.

`LVDefs_plat.h` contains basic constants and macros that the C Generator needs to generate C code. This file provides the mapping between generic C function calls that the C Generator generates and platform-specific, run-time functions. `LVDefs_plat.h` also contains platform definitions of data types for the compiler.

To properly define data types, you must know the compiler definitions of those data types. To define the generic function calls, you must know how the target SDK implements these function calls. `LVDefs_plat.h` defines the data types and function calls correctly for a Cygwin target.

12. Close `LVDefs_plat.h`.
13. Open and review `LVSysIncludes.h`, which is located in the `labview\CCodeGen\include\platform\cygwin-gcc` directory.

`LVSysIncludes.h` contains platform-specific header files, toolchain-specific header files, and standard C header files. Most files in the `labview\CCodeGen` directory include `LVSysIncludes.h`.

14. Close `LVSysIncludes.h`.

Running the GNU Make Utility to Create an Executable

You use the GNU Make utility to build applications and libraries automatically from the generated C code. Before you can run the `Makefile` in the directory that contains the generated C code, you must set up the required toolchain.

Complete the following steps to set up the toolchain and run the GNU Make utility.

1. Use the Cygwin command line to verify that GCC and the GNU Make utility are on the PATH.
2. To create `GCD.exe`, call the `make GCD.exe` command from the command line.

Running and Calling the Generated C Code

You now can run `GCD.exe` to test the exported function. When you run the application, `main.c` calls `GreatestCommonDivisor(12, 15)`. The expected output of the exported function is 3.

EK-LM3S8962 Application

The C Generator installs a μ Vision project (`.Uv2`) you can use to build the generated C code in an ARM application. Complete the following steps to use the generated C code in an ARM application built in Keil μ Vision for a Luminary Micro EK-LM3S8962 target.

1. In Keil μ Vision, open `labview\examples\CGenerator\Tutorial\Keil - ARM LM3S8962\example.Uv2`.
2. Expand the **LM3S8962** target in the **Project Workspace** window.
3. Expand the **Generated Files** folder. Notice that folder includes `GCD.c` and `NI_Gmath_lvlib_Gcd.c`, which are the generated C files.
4. Right click the **LM3S8962** target in the **Project Workspace** window and select **Options for Target** from the shortcut menu.
5. On the **Target** tab, notice that **Operating system** is set to **Nothing**. Do not set this value to RTX, which is incompatible with the generated C code.
6. On the **Target** tab, ensure that the **Use MicroLIB** option does not contain a checkmark. This option is incompatible with the generated C code.

7. On the **C/C++** tab, review the value of the **Include Paths** option. The include paths are relative to the location of `example.Uv2`.
8. Right-click the **LM3S8962** target in the **Project Workspace** window and select **Build target** from the shortcut menu to build the target application.

Running and Calling the Generated C Code

You now can run the application to test the exported function. Complete the following steps to run the application.

1. In Keil μ Vision, select **Debug»Start/Stop Debug Session** to start the debugger.
2. Click the **Run** button to execute the program.



Note If you want to run the application in release mode, click the **Download to Flash** toolbar button in μ Vision to download the application to flash memory. After the download is complete, press the RESET button on the board to start the application.

The program calls `GreatestCommonDivisor(12, 18)`. After you click the **Run** button, the LED blinks six times to return the greatest common divisor of 12 and 18.

Where to Go from Here

The following documents contain information that you might find helpful as you use the LabVIEW C Generator:

- *LabVIEW Help*, available by selecting **Help»LabVIEW Help** in LabVIEW, provides information about LabVIEW programming, step-by-step instructions for using LabVIEW, and reference information about LabVIEW VIs, functions, palettes, menus, and tools. Refer to the **C Generator** book on the **Contents** tab of the *LabVIEW Help* for information specific to the C Generator and applications you create. The *LabVIEW Help* uses `(C Generator)` in the index to indicate topics specific to the C Generator.
- Context help provides brief descriptions of VIs and functions with a link to the complete reference for a VI or function. Select **Help»Show Context Help** to open the **Context Help** window.
- *LabVIEW C Generator Readme*—Use this file to learn important last-minute information about the C Generator, including known issues. Open the readme file by selecting **Start»All Programs»National Instruments»LabVIEW»Readme** and opening `readme_CGenerator.html`.
- LabVIEW PDFs—In addition to this document, the *Getting Started with LabVIEW* manual, *LabVIEW Quick Reference Card*, *LabVIEW Release Notes*, and *LabVIEW Upgrade Notes* are available as PDFs by selecting **Start»All Programs»National Instruments»LabVIEW»LabVIEW Manuals**.
- Documentation for any third-party tools, the CPU, and processor with which you plan to use the generated C code.
- C Generator Examples—Use the C Generator examples as a starting point for developing VIs to export for C code generation. You can modify an example, or you can copy and paste from one or more examples into a VI that you create. Use the NI Example Finder, available by selecting **Help»Find Examples**, to browse or search the example VIs. You also can browse the examples by navigating to the `labview\examples\CGenerator` directory.

LabVIEW, National Instruments, NI, ni.com, the National Instruments corporate logo, and the Eagle logo are trademarks of National Instruments Corporation. Refer to the *Trademark Information* section at ni.com/trademarks for other National Instruments trademarks. ARM, Keil, and μ Vision are trademarks or registered trademarks of ARM Ltd or its subsidiaries. Other product and company names mentioned herein are trademarks or trade names of their respective companies. For patents covering National Instruments products/technology, refer to the appropriate location: **Help»Patents** in your software, the `patents.txt` file on your media, or the *National Instruments Patent Notice* at ni.com/patents.