Getting Started with the LabVIEW™ Embedded Development Module

Evaluation Edition

The LabVIEW Embedded Development Module enables you to port LabVIEW to new targets, including non-NI hardware. Porting LabVIEW involves creating a target, customizing the build process, porting the run-time environment, and customizing the editing environment for the new target.

The LabVIEW Embedded Development Module Evaluation Edition shows you how to create VIs and applications after you or someone else ports LabVIEW to an embedded target.

Contents

NATIONAL INSTRUMENTS™

Evaluation Edition Limitations

The LabVIEW Embedded Development Module Evaluation Edition is a time-based evaluation. It has the following limitations:

- Front panel and block diagram watermarks on all VIs you create.
- VIs you run on Windows automatically stop running after five minutes.
- This evaluation stops working 60 days after you install the evaluation or December 31, 2006, whichever occurs first.
- Any applications you build with this evaluation stop working after December 31, 2006.
- All applications you build with this evaluation contain an `Evaluation version of LabVIEW 7.1 Embedded Edition` message when the application begins running.
- This evaluation does not include the source code to the run-time library.
- You cannot create new targets or download applications to your own target.

Evaluation Example Target and Toolchain

This evaluation edition includes one example target—the Unix - evaluation target. National Instruments chose this example target because it does not require any additional hardware.

When you buy the LabVIEW Embedded Development Module, you receive several example targets to use as a starting point when you create new embedded targets. Refer to the *LabVIEW Embedded Development Module Porting Guide*, available by selecting **Start»All Programs» National Instruments»LabVIEW 7.1 Embedded Edition» LabVIEW Manuals** and opening `EMB_Porting_Guide.pdf`, for more information about the different example targets and their implementations. The Unix - evaluation target uses the Cygwin with gcc toolchain, which is included in this evaluation.

System Requirements

The LabVIEW Embedded Development Module Evaluation Edition has the following requirements:

- A desktop computer with Windows 2000/XP
- Cygwin with gcc (included)

Installing and Uninstalling

The LabVIEW Embedded Development Module Evaluation Edition uses LabVIEW 7.1 Embedded Edition, which is a special edition of LabVIEW 7.1 that installs in a separate directory and does not interfere with LabVIEW 7.1 or later Base, Full, or Professional development systems.

Installing

Complete the following steps to install the LabVIEW Embedded Development Module Evaluation Edition.

1. Log on as an administrator or as a user with administrator privileges.
2. Insert the LabVIEW Embedded Development Module Evaluation Edition CD and follow the instructions that appear on the screen.

Uninstalling

To uninstall the LabVIEW Embedded Development Module Evaluation Edition, remove the following using Add or Remove Programs:

- NI LabVIEW 7.1 Embedded Edition - Unix Target
- NI LabVIEW 7.1 Embedded Evaluation Edition - Unix Target

About This Manual

This manual includes two tutorials that demonstrate using LabVIEW with embedded targets. The first tutorial shows instrumented debugging and front panel connectivity. The second tutorial shows how to use an Inline C Node, which you can use to add C code to an embedded VI, and Elemental I/O Nodes, which provide a common interface for accessing I/O resources.

The *Getting Started* section shows you how to select an execution target, verify the palette view, and create a LabVIEW Embedded Project. Complete the *Getting Started* section prior to each tutorial.

Getting Started

The first step in creating a VI or application is selecting the execution target, which determines where the application runs, and verifying you are using the correct palette view, which contains the controls, indicators, VIs, and functions you can use for that target.

Selecting the Execution Target

The execution target determines where you run the VI or application. Complete the following steps to launch LabVIEW Embedded Edition and select the Unix - evaluation execution target.

1. Launch LabVIEW 7.1 Embedded Edition.
2. In the **LabVIEW** dialog box, shown in Figure 1, select **Unix - evaluation** from the **Execution Target** pull-down menu.

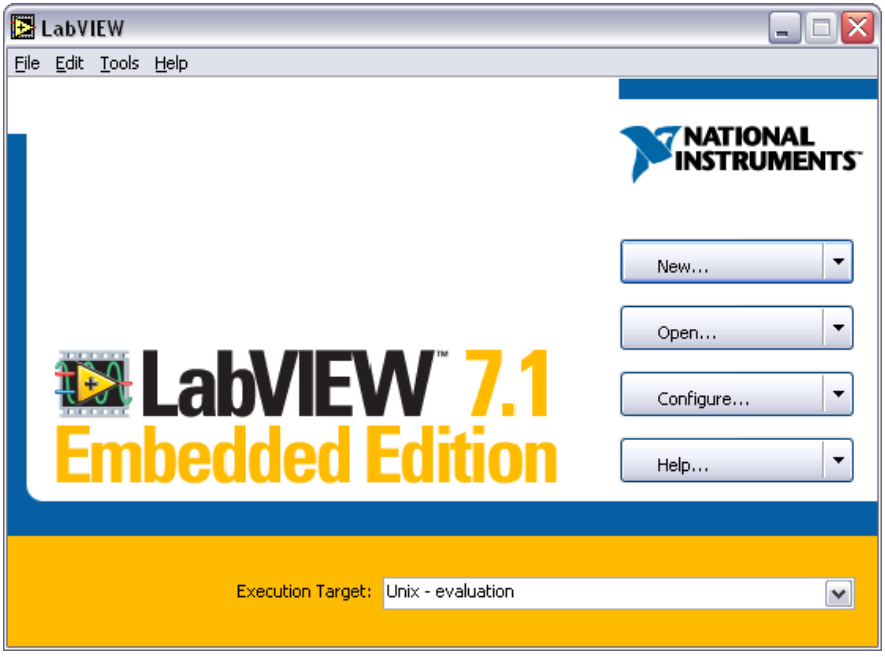

Figure 1. LabVIEW Dialog Box

3. Click the down arrow next to the **New** button and select **Blank VI** from the shortcut menu.

The front panel and the block diagram open. The front panel, or user interface, appears with a gray background and includes controls and indicators. The block diagram appears with a white background and includes VIs, functions, and structures that control the front panel.

Changing the Palette View

The **Controls** palette is available only on the front panel and contains controls and indicators you use to build the user interface. The **Functions** palette is available only on the block diagram and contains the VIs and functions you use to build a VI. The objects on the palettes depend on the current palette view.

You can create and edit palette views to remove VIs and functions from the palette that a target does not support. For example, you can remove the TCP/IP VIs for targets without Ethernet connectivity. The LabVIEW Embedded Development Module includes a generic Embedded palette view, which contains the VIs and functions the LabVIEW C Code Generator, which generates the C code from the block diagram, supports. The generic Embedded palette view is a subset of the standard advanced palette view.

Complete the following steps to change the palette view to the Embedded palette.

Note This tutorial changes the palette view from the **Functions** palette, which is on the block diagram. You also can change the palette view from the **Controls** palette, which is on the front panel.

1. If the **Functions** palette, shown in Figure 2, is not visible on the block diagram, right-click any blank space on the block diagram to display a temporary version of the **Functions** palette. Click the thumbtack in the upper left corner of the **Functions** palette to pin the palette so it is no longer temporary. You can place the **Functions** palette anywhere on the screen.

Figure 2. Functions Palette–Default Advanced View

Tip You also can select **Window»Show Functions Palette** to display the palette.

2. Click the **Options** button, shown at left, on the **Functions** palette to open the **Controls/Functions Palettes** page of the **Options** dialog box.

3. Select **Embedded** from the **Palette View** pull-down menu as shown in Figure 3.

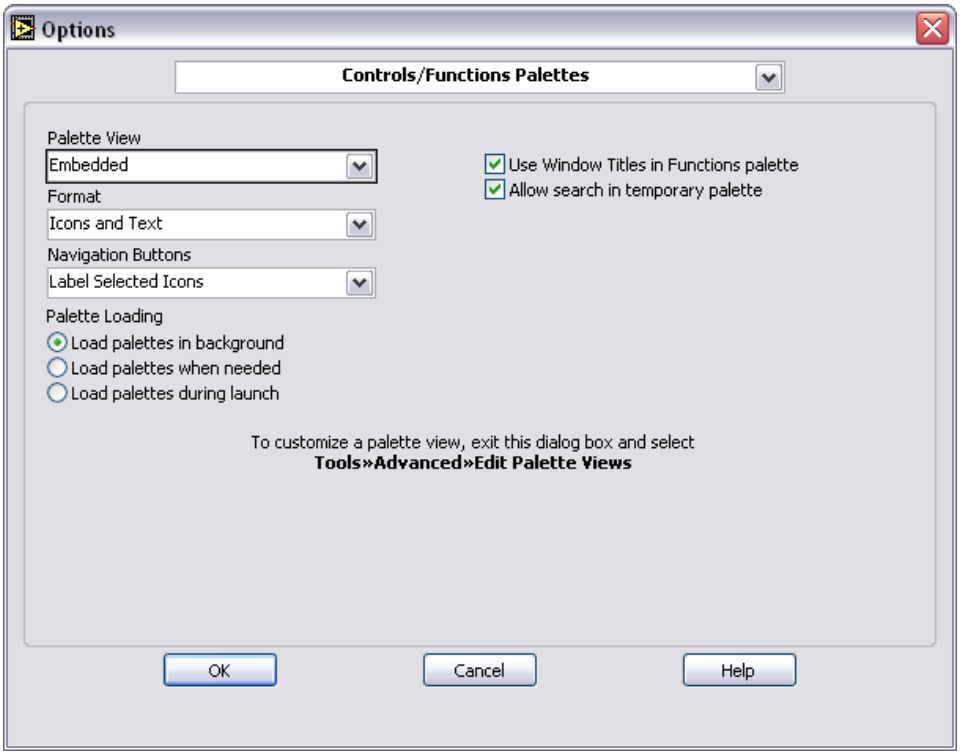

Figure 3. Changing the Palette View

4. Click the **OK** button. LabVIEW loads the Embedded palette.

Creating the LabVIEW Embedded Project

Embedded VIs use a LabVIEW Embedded Project, which you create and configure using the **Embedded Project Manager** window, to manage groups of VIs, select build options, build, and run embedded applications. LabVIEW Embedded Project files have a `.lep` file extension. LEP files contain target-specific build options and other information necessary for the LabVIEW C Code Generator to generate C code from the VIs.

Complete the following steps to create a new embedded project.

1. Select **Tools»Embedded Project Manager** to open the **Embedded Project Manager** window as shown in Figure 4.

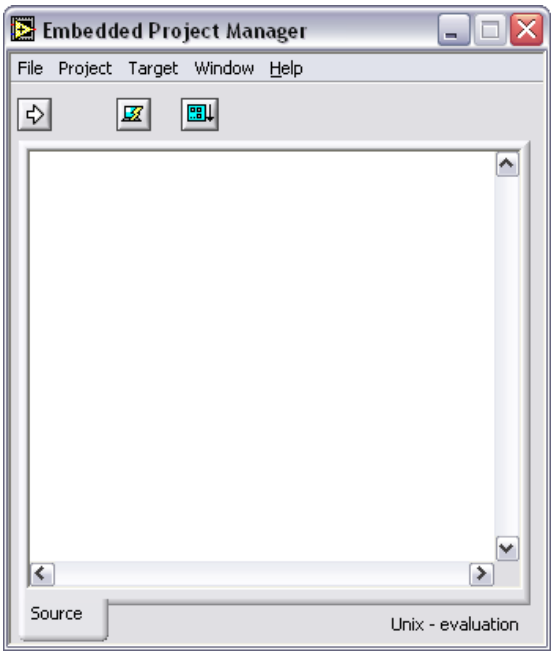

Figure 4. Embedded Project Manager Window

2. Select **File»New** in the **Embedded Project Manager** window to open the **New** dialog box.
3. On the **Projects** tab, enter a name for the project in the **Project name** text box.

4. Enter a path in the **Project location** path control or browse to the location where you want to save the LEP file as shown in Figure 5.

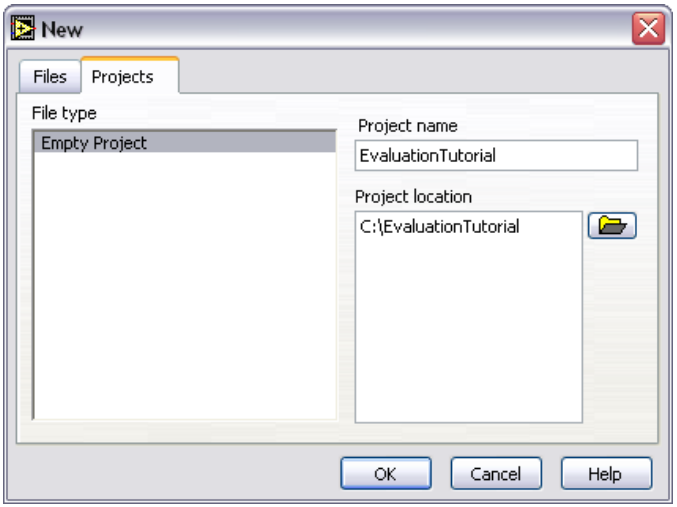

Figure 5. Creating the LabVIEW Embedded Project

5. Click the **OK** button.

Adding a New VI to the Embedded Project

You can add new or existing VIs to LEP files. You also can add external `.c` files or `.lib` files to LEP files.

Complete the following steps to create a new VI and add it to the embedded project.

1. Select **Project»Add to Project»New** to open the **New** dialog box. You also can right-click **Source Files** and select **Add New** from the shortcut menu.
2. Select **Blank VI** from the **File type** list.
3. Enter a name for the VI in the **Filename** text box.
4. Enter a path in the **Location** path control or browse to the location where you want to save the VI as shown in Figure 6. LabVIEW saves the new VI in the same directory as the LEP file by default. You can save the VI in a different directory, but National Instruments recommends saving the LEP file and the top-level VI in the same directory.

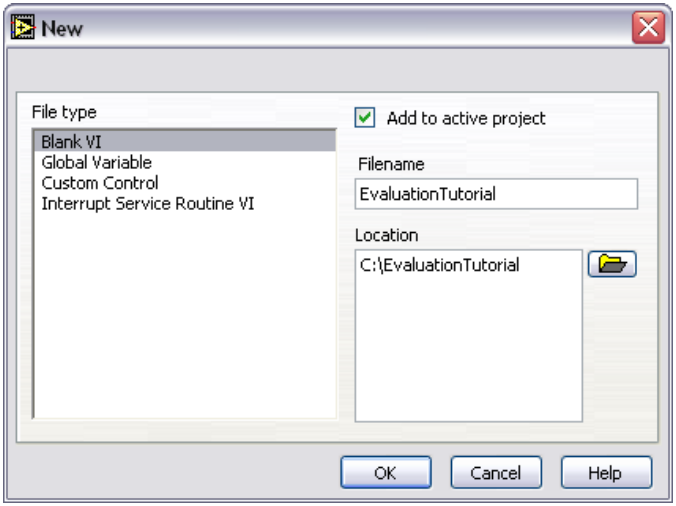

Figure 6. Adding a Blank VI to the Embedded Project

5. Click the **OK** button.

The front panel opens and the VI appears under **Source Files** in the **Embedded Project Manager** window as shown in Figure 7.

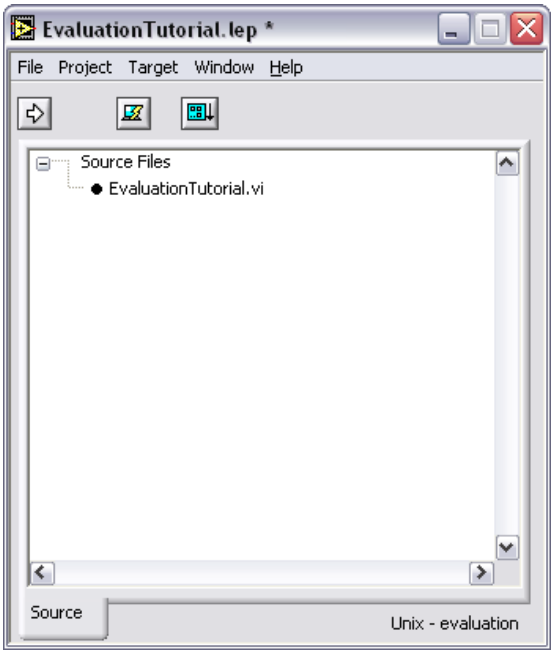

Figure 7. Embedded Project Manager Window After Adding a VI

Refer to *Tutorial 1—Instrumented Debugging* to see how to communicate with an embedded application and provide front panel connectivity for debugging.

Refer to *Tutorial 2—Elemental I/O and Inline C Node* to see how to add inline C code to an embedded VI.

Tutorial 1—Instrumented Debugging

This tutorial shows how to create a VI that uses instrumented debugging to communicate with an embedded application running on an embedded target, which is the Unix - evaluation target for this evaluation, and provide front panel connectivity and block diagram breakpoints. Instrumented debugging inserts additional code for debugging purposes.

Instrumented debugging occurs through synchronization and data-transfer routines in the generated C code of an embedded VI. These routines use an underlying communication layer, such as serial or TCP/IP, which you must provide for your embedded targets and LabVIEW. You implement the instrumented debugging layer for LabVIEW with plug-in VIs. You must implement instrumented debugging on the target with C functions. The Unix - evaluation target in this tutorial communicates over TCP/IP for debugging. This tutorial shows how you can use instrumented debugging after it has been implemented for a target.

Refer to the *Getting Started* section for instructions on creating the LabVIEW Embedded Project and VI, which is required for embedded targets and this tutorial.

Note The completed VI is located in `labview embedded\examples\embedded evaluation\Debug_Tutorial.vi`.

Designing the Front Panel

The front panel is the user interface of the VI for desktop VIs and applications you run on Windows. For embedded applications, you use the front panel for interactive debugging.

You build the front panel with controls and indicators, which are the interactive input and output terminals of the VI, respectively. Controls are knobs, push buttons, dials, and other input mechanisms. Indicators are graphs, LEDs, and other output displays. Controls supply data to the block diagram of the VI. Indicators display data the block diagram acquires or generates. When you place a control or indicator on the front panel, a corresponding terminal appears on the block diagram.

The front panel for this tutorial uses a waveform graph to display a signal that you manipulate using front panel controls. Complete the following steps to create the front panel for this tutorial.

Note Refer to Figure 8 to see the completed front panel for this tutorial.

1. Place the following controls and indicators on the block diagram:
   - One waveform graph located on the **Controls»Graph** palette.
   - Two numeric controls located on the **Controls»Numeric** palette.

Tip If you cannot find the object you are looking for, click the **Search** button on the **Controls** or **Functions** palette toolbar. Type the text for which you want to search. LabVIEW searches as you type and displays any matches in the search results text box.

2. Change the label of one numeric control to **Noise** and the label of the other numeric control to **Signal** as shown in Figure 8.

Tip Double-click to select a single word in a label. Triple-click to select the entire label.

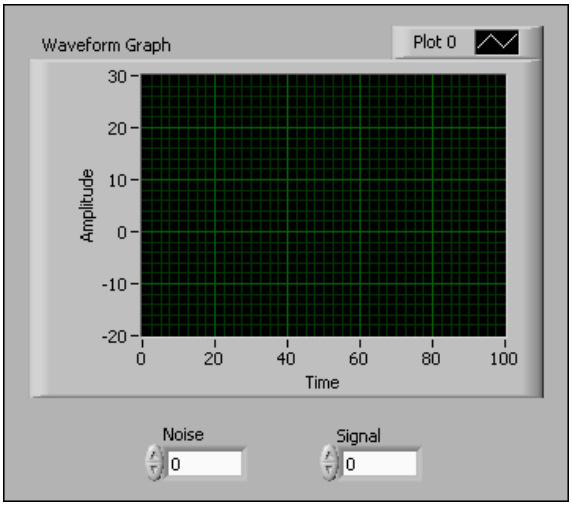

Figure 8. Designing the Front Panel

3. Change the default values for the numeric controls.
   a. Right-click the **Noise** numeric control and select **Properties** from the shortcut menu to open the **Numeric Properties** dialog box.
   b. Click the **Data Range** tab.
   c. Enter `1` in the **Default value** box.

d. Click the **OK** button.

e. Repeat steps a–d and change the default value of the **Signal** numeric control to `10`.

4. Save the VI.

Creating the Block Diagram

After you build the front panel, you add the graphical code to the block diagram. Objects on the block diagram include terminals and nodes, which include VIs, functions, structures, and so on. Nodes are analogous to statements, operators, functions, and subroutines in text-based programming languages. Terminals are objects or regions on a node through which data passes. The color and symbol of each terminal indicate the data type of the corresponding control or indicator. Front panel objects you add appear as terminals on the block diagram. Constants are terminals on the block diagram that supply fixed data values to the block diagram.

You create block diagrams by connecting the objects with wires. Each wire has a single data source, but you can wire the data source to many VIs and functions that read the data, similar to passing required parameters in text-based programming languages. Wires are different colors, styles, and thicknesses depending on their data types. Refer to the *Control and Indicator Data Types* topic in the *LabVIEW Help*, which is available by selecting **Help»VI, Function, & How-To Help**, for more information about the visual appearance of different data types.

The block diagram in this tutorial multiplies the value of the **Noise** control by a random number and adds the output to a sine wave, which is multiplied by the value of the **Signal** control. The waveform graph then displays the output.

Complete the following steps to create the block diagram.

Note Refer to Figure 12 to see the completed block diagram for this tutorial.

1. Switch to the block diagram by pressing the <Ctrl-E> keys.

Tip You also can click the block diagram if it is visible or select **Window»Show Block Diagram**.

2. Select **Help»Show Context Help** to display the **Context Help** window. The **Context Help** window displays basic information about LabVIEW objects when you move the cursor over each object.

Tip You also can press the <Ctrl-H> keys to open and close the **Context Help** window.

Generating the Noise Component of the Signal

The noise component is the value of the **Noise** control multiplied by a random number.

1. Place a Random Number (0-1) function located on the **Functions» Numeric** palette on the block diagram.

Tip You can click the **Search** button on the palette toolbar to perform text-based searches for any control, VI, or function on the **Controls** and **Functions** palettes.

2. Place two Multiply functions located on the **Functions»Numeric** palette on the block diagram.
3. Wire the **number (0-1)** output of the Random Number (0-1) function to the **x** input of one of the Multiply functions.
4. Wire the **Noise** numeric control to the **y** input of the same Multiply function you used in step 3 to multiply the noise value by a random number as shown in Figure 9.

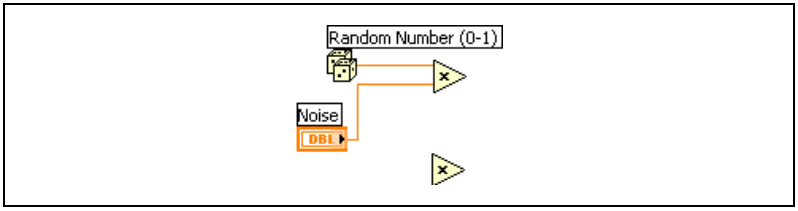

Figure 9. Generating the Noise Component of the Signal

Creating the Sine Wave

The sine wave you create is multiplied by the value of the **Signal** control.

1. Wire the **Signal** numeric control to the **x** input of the other Multiply function.
2. Place the Sine function located on the **Functions»Numeric» Trigonometric** palette on the block diagram.
3. Wire the **sin(x)** output of the Sine function to the **y** input of the Multiply function you used in step 1.
4. Place an Add function located on the **Functions»Numeric** palette on the block diagram.
5. Wire the **x*y** output of one of the Multiply functions to the **x** input of the Add function. Wire the output of the other Multiply function to the **y** input of the Add function.
6. Place a Divide function located on the **Functions»Numeric** palette on the block diagram.

7. Right-click the **x** input of the Divide function and select **Create» Constant** from the shortcut menu. Enter `5` to scale the frequency of the sine wave.
8. Wire the **x/y** output of the Divide function to the **x** input of the Sine function as shown in Figure 10.

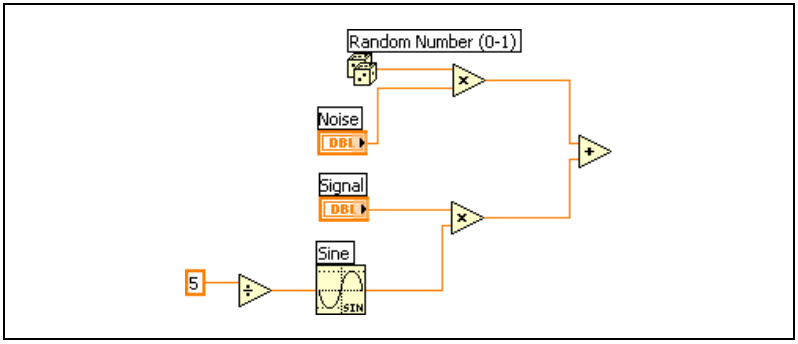

Figure 10. Creating the Sine Wave

Creating the For Loop

The For Loop determines how many times the code inside of the loop executes. For this tutorial, the code executes 128 times.

1. Place a For Loop located on the **Functions»Structures** palette around the controls and functions on the block diagram. Move the Waveform Graph indicator by dragging it outside of the For Loop.
2. Wire the iteration (i) terminal of the For Loop to the **y** input of the Divide function.
3. Right-click the count (N) terminal and select **Create Constant** from the shortcut menu. Enter `128` to execute the code 128 times. This tutorial uses 128 to keep the data set manageable, but you can experiment with different values.
4. Wire the **x+y** output of the Add function to the Waveform Graph indicator outside of the For Loop as shown in Figure 11.

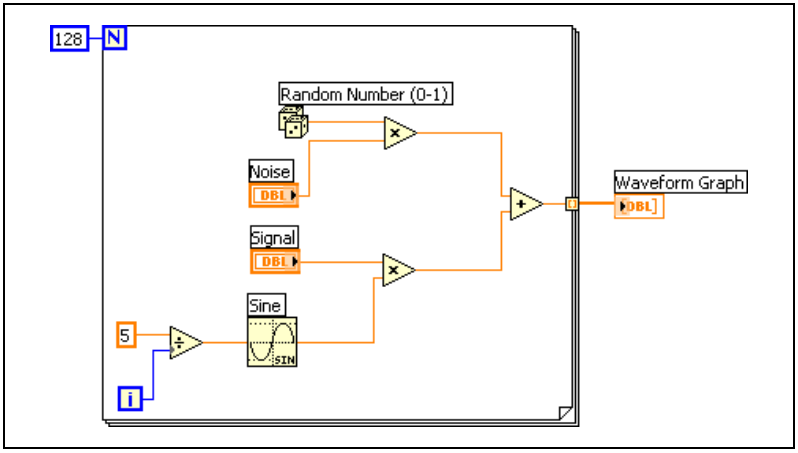

Figure 11. Creating the For Loop

Creating the Infinite Outer While Loop

Embedded applications often use an infinite outer While Loop to remove any dependencies to the host computer.

1. Place a While Loop located on the **Functions»Structures** palette around the For Loop and Waveform Graph indicator on the block diagram. Right-click the conditional terminal in the lower right corner of the While Loop and select **Create Constant** from the shortcut menu. The default Boolean constant in the While Loop is FALSE.
2. (Optional) Place a Wait Until Next ms Multiple function located on the **Functions»Time, Dialog & Error** palette inside the While Loop. Right-click the **millisecond multiple** input and select **Create» Constant** from the shortcut menu. Enter `50` to wait 50 milliseconds as shown in Figure 12.

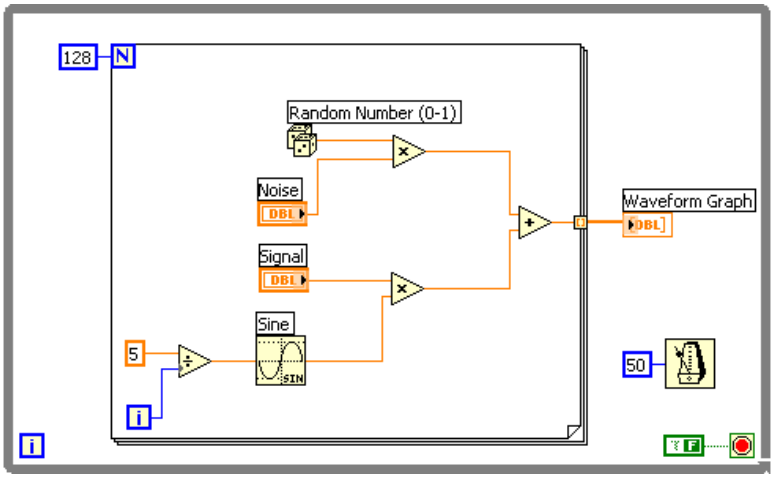

Figure 12. Creating the Infinite Outer While Loop

Building the Embedded VI into an Embedded Application

Before you can build an embedded VI into an embedded application, you must select the build options in the **Embedded Project Manager** window. Build options are target-specific so different targets may have different options.

Complete the following steps to configure the build options for this VI.

1. Select **Target»Build Options** in the **Embedded Project Manager** window.
2. Under **Build options**, select **Release** from the **Build configuration** drop-down list.
3. Under **Debug options**, place a checkmark in the **Generate debug info** checkbox as shown in Figure 13.

   Generating debug information adds extra code you need to debug the embedded application over various protocols, such as TCP/IP or RS232. The extra code is in the form of C function calls that update the application state and communicate the state to the host computer for display. Generating debug information usually results in a 25%–40% increase in code size.
4. (Optional) Click the **Help** button for a description of the common build options.

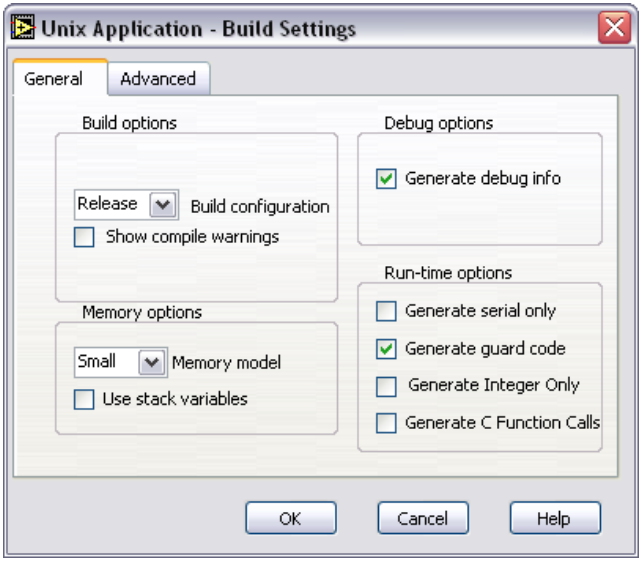

Figure 13. Configuring the Build Options

5. Click the **OK** button.
6. Save the VI.
7. Select **Target»Build** in the **Embedded Project Manager** window to build the embedded VI into an embedded application using the build options you configured. The LabVIEW C Code Generator generates C code from the block diagram you created and compiles the C Code using the toolchain you supply. For this tutorial, the compiler is gcc. The generated C code is located in a folder in the same directory where you save the VI.

Note Although you can directly edit the C code, the LabVIEW C Code Generator overwrites any edits you make to the generated C code the next time you generate the C code. National Instruments does not recommend directly editing the generated C code. Instead, use an Inline C Node located on the **Functions»Structures** palette to directly affect the generated C code.

Downloading and Debugging the Embedded Application

After you build the embedded VI into an embedded application, you can download and debug the embedded application. Debugging an embedded application using LabVIEW involves using a real-time connection between LabVIEW and the embedded target.

Complete the following steps to download and debug the embedded application.

1. Select **Target»Debug** in the **Embedded Project Manager** window to start the embedded application on the Unix - evaluation target and begin an instrumented debugging session. The console window appears.

   The LabVIEW C Code Generator includes additional data structures and function calls for instrumented debugging to synchronize with LabVIEW and uses a communication link, such as TCP, as a debug channel. In this tutorial, the embedded application is updating the front panel through TCP/IP on localhost (IP address 127.0.0.1).

   The embedded application begins running on the Unix - evaluation target and the graph begins updating on the front panel. You can minimize the console window if you cannot see the front panel.

Note Selecting **Target»Run** in the **Embedded Project Manager** window runs the embedded application on the target, but the application does not establish a connection with the front panel on the host computer. You must debug the application for interactions with the front panel on the host computer to affect the embedded application running on the target.

2. On the front panel, change the values of the **Noise** and **Signal** numeric controls to affect the execution of the application.
3. On the block diagram, right-click the wire connecting the output of the Random Number (0-1) function to the **x** input of the Multiply function and select **Set Breakpoint** from the shortcut menu. A red dot appears on the wire to signify a breakpoint.

4. When you set a breakpoint on a wire, execution pauses after data passes through the wire. The VI pauses when you reach a breakpoint during execution and the **Pause** button, shown at left, appears red.

5. Single-step through the VI to view each action of the VI as the corresponding embedded application runs. Click the **Step Over** button, shown at left, to start single-step mode.

Tip Move the cursor over the **Step Over** button to view a tip strip that describes the next step if you click the button.

6. Click the **Abort Execution** button, shown at left, to stop the embedded application running on the Unix - evaluation target.
7. Press the <Enter> key in the console window to terminate the Unix - evaluation application session.

This tutorial shows how to use instrumented debugging and front panel connectivity to debug an embedded application. You or someone else must implement debugging for a new target, which this tutorial does not show. Refer to Chapter 13, *Debugging*, of the *LabVIEW Embedded Development Module Porting Guide*, which is available by selecting **Start»All Programs»National Instruments»LabVIEW 7.1 Embedded Edition»LabVIEW Manuals** and opening `EMB_Porting_Guide.pdf`, for information about implementing debugging for a new target.

Tutorial 2—Elemental I/O and Inline C Node

Elemental I/O is a flexible, user-defined way to create single-point I/O for LabVIEW embedded targets. Elemental I/O Nodes are portable across many targets. Although the I/O implementation might change among targets, you do not need to redevelop the application for another target. Use Elemental I/O Nodes to create block diagrams that represent algorithms that you can reuse on many platforms.

The Inline C Node provides a way to add C or assembly code to the block diagram. Use the Inline C Node for short blocks of code that you cannot easily implement in VIs. The Inline C Node contains support and functionality for low-level programming and header files without the overhead of a function call.

This tutorial show how to use an elemental I/O node and the Inline C Node. It is not meant to show a typical embedded application. This tutorial does not include a front panel because the tutorial does not include debugging.

Refer to the *Getting Started* section for instructions on creating the LabVIEW Embedded Project and VI, which is required for embedded targets and this tutorial.

Note The completed VI is located in `labview embedded\examples\embedded evaluation\IO_Tutorial.vi`.

Creating the Block Diagram

The block diagram is where you add the graphical code. Objects on the block diagram include terminals and nodes, which include VIs, functions, structures, and so on. Nodes are analogous to statements, operators, functions, and subroutines in text-based programming languages. Terminals are objects or regions on a node through which data passes. The color and symbol of each terminal indicate the data type of the corresponding control or indicator. Front panel objects you add appear as terminals on the block diagram. Constants are terminals on the block diagram that supply fixed data values to the block diagram.

You create block diagrams by connecting the objects with wires. Each wire has a single data source, but you can wire it to many VIs and functions that read the data, similar to passing required parameters in text-based programming languages. Wires are different colors, styles, and thicknesses depending on their data types.

This block diagram in this tutorial uses a Digital Input elemental I/O node to generate a random Boolean value and sends the output to the console window. For most embedded targets, the Digital Input elemental I/O node maps to a general purpose I/O pin. Also, because this tutorial is not meant to show a typical embedded application, the block diagram does not include an infinite outer While Loop, which Tutorial 1 uses and is common in embedded applications.

Complete the following steps to create the block diagram.

Note Refer to Figure 17 to see the completed block diagram for this tutorial.

1. Switch to the block diagram by pressing the <Ctrl-E> keys.

Tip You also can click the block diagram if it is visible or select **Window»Show Block Diagram**.

2. Select **Help»Show Context Help** to display the **Context Help** window. The **Context Help** window displays basic information about LabVIEW objects when you move the cursor over each object.

Tip You also can press the <Ctrl-H> keys to open and close the **Context Help** window.

Configuring the Elemental I/O Node

The elemental I/O node in this tutorial generates a random Boolean value.

1. Place a Digital Input elemental I/O node located on the **Functions» Elemental I/O** palette on the block diagram.
2. Double-click the elemental I/O node to configure it. The **Configure Elemental I/O** dialog box appears as shown in Figure 14. The configuration dialog boxes are target-specific.

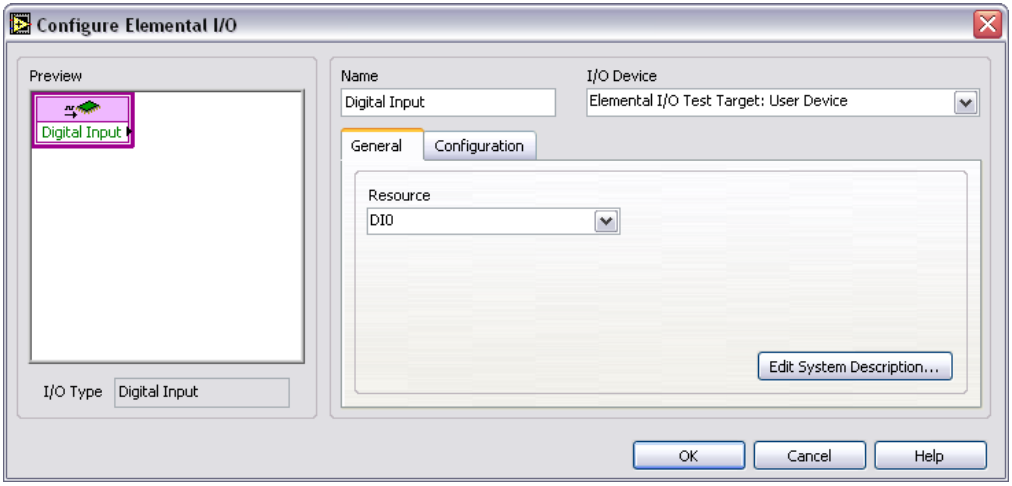

Figure 14. Configuring the Elemental I/O Node

3. Click the **Configuration** tab.
4. Select **Random** from the **Mode** drop-down list to return a random Boolean value.
5. Click the **OK** button.

Adding an Inline C Node and Case Structure

The Inline C Node you create uses the `printf` function to write the enumerated iteration and the word `YES` or `NO`, depending on the Boolean value of the Digital Input elemental I/O node, to the console window.

1. Place an Inline C Node located on the **Functions»Structures** palette on the block diagram. Use Inline C Nodes to add C code to the block diagram. You can add any C code, including assembly directives and `#defines`, that syntactically is between the curly braces in a C file.

Note Do not use `return` or `exit` statements in Inline C Nodes.

2. Right-click the left border of the Inline C Node and select **Add Input** from the shortcut menu. Name the input `Iteration`.
3. Enter the following in the Inline C Node as shown in Figure 15:

```
printf("Iteration %d; YES\n", Iteration);
```

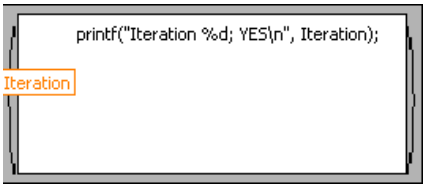

Figure 15. Inline C Node

4. Place a Case structure located on the **Functions»Structures** palette around the Inline C Node.
5. Wire the output of the Digital Input elemental I/O node to the selector terminal of the Case structure as shown in Figure 16.

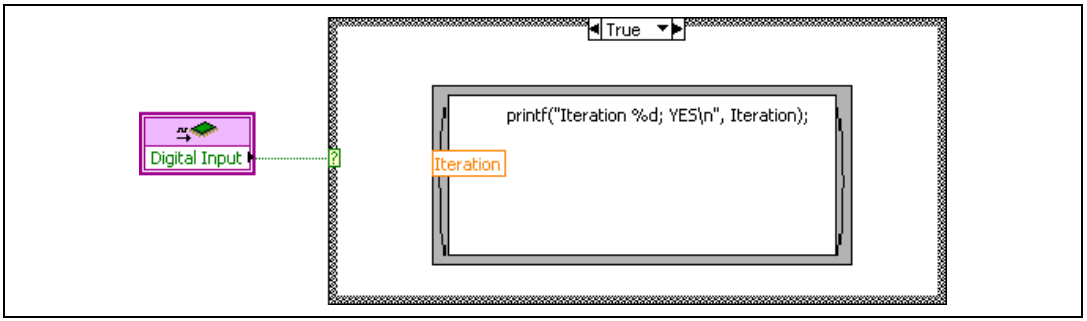

Figure 16. Wiring the Elemental I/O Node to the Case Structure

6. Copy the Inline C Node and select the False case. Paste the Inline C Node in the False case.
7. Change the code in the Inline C Node in the False case to the following:

```
printf("Iteration %d; NO\n", Iteration);
```

Creating the For Loop

The For Loop determines how many times the code inside the loop executes. For this tutorial, the code executes 20 times.

1. Place a For Loop located on the **Functions»Structures** palette around the code.
2. Right-click the count (N) terminal and select **Create Constant** from the shortcut menu. Enter `20` to execute the code 20 times. You can experiment with this number.
3. Place an Increment function located on the **Functions»Numeric** palette inside the For Loop.
4. Wire the iteration (i) terminal of the For Loop to the input of the Increment function.

5. Wire the output of the Increment function to the **Iteration** input of the Inline C Node. Notice that the color of the wire and the input changes from orange, which represents a floating-point numeric to blue, which represents an integer numeric.

Tip Hover the mouse over the wire to see the data type in the **Context Help** window, which is available by pressing the <Ctrl-H> keys.

6. Repeat step 5 for the True case of the Case structure.
7. (Optional) Place a Wait Until Next ms Multiple function located on the **Functions»Time, Dialog & Error** palette inside the For Loop. Right-click the input of the Wait Until Next ms Multiple function and select **Create»Constant** from the shortcut menu. Enter `250` to wait 250 milliseconds.
8. Save the VI.

The VI should look similar to Figure 17.

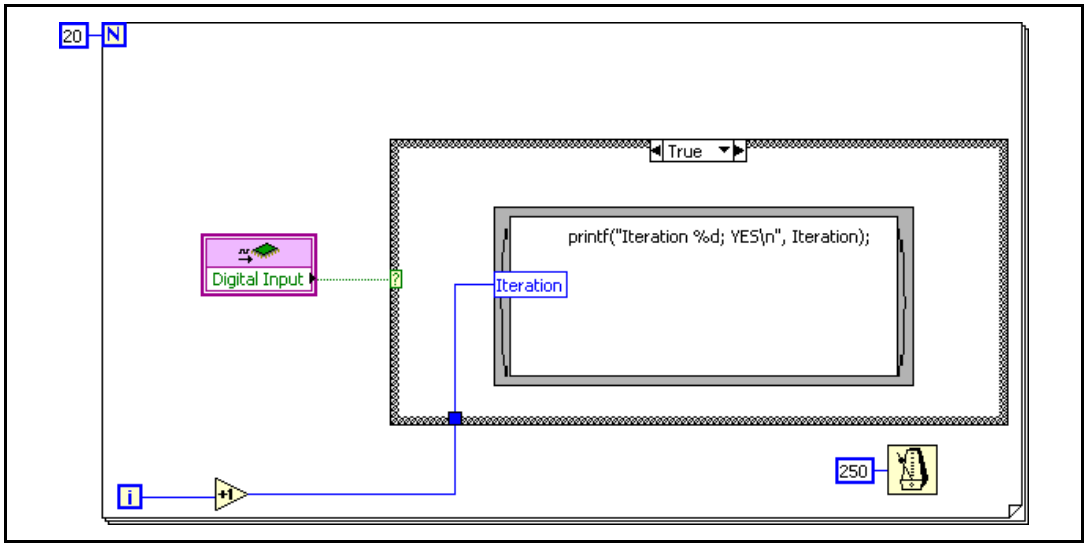

Figure 17. Completed VI

Building the Embedded VI into an Embedded Application

Before you can build an embedded VI into an embedded application, you must configure the build options in the **Embedded Project Manager** window. Build options are target-specific so different targets may have different options. If you do not configure the build options, LabVIEW uses the default build options for the current execution target.

Use the default build options for this tutorial. Select **Target»Build Options** in the **Embedded Project Manager** window to see the default build options for the Unix - evaluation target. Click the **Help** button for a description of the most common build options. Save the VI.

Select **Target»Build** in the **Embedded Project Manager** window to build the embedded VI into an embedded application. The LabVIEW C Code Generator generates C code from the block diagram you create. The generated C code is located in a folder in the same directory where you save the VI.

Note Although you can directly edit the C code, the LabVIEW C Code Generator overwrites any edits you make to the generated C code the next time you generate the C code. National Instruments does not recommend directly editing the generated C code. Instead, use an Inline C Node located on the **Functions»Structures** palette to directly affect the generated C code.

Downloading and Running the Embedded Application

After you build the embedded VI into an embedded application, you can download and run the embedded application. Select **Target»Run** in the **Embedded Project Manager** window to start the embedded application on the Unix - evaluation target.

The `printf` function in the Inline C Node sends the output to the console window. Press the <Enter> key to terminate the Unix - evaluation application session.

This tutorial shows how to use Elemental I/O and the Inline C Node. You or someone else must implement Elemental I/O before you can use it with an embedded target. Refer to Chapter 11, *Elemental I/O*, of the *LabVIEW Embedded Development Module Porting Guide*, which is available by selecting **Start»All Programs»National Instruments»LabVIEW 7.1 Embedded Edition»LabVIEW Manuals** and opening `EMB_Porting_Guide.pdf`, for information about implementing Elemental I/O for a new target. Refer to the *Inline C Node* topic in the *LabVIEW Help*, which is available by selecting **Help»VI, Function, & How-To Help**, for more information about the Inline C Node.

Where To Go from Here

Visit `ni.com/embedded` for the following:

- Technical tutorials
- Additional product documentation
- Complementary partner tools and targets
- Links to the developer community
- Information on pricing and ordering

371853A-01 Nov05