Document Type: Tutorial
NI Supported: Yes
Publish Date: Apr 9, 2008

Using LabWindows/CVI DLLs in LabVIEW Real-Time Applications

Overview

LabWindows/CVI extends the functionality of LabVIEW Real-Time by allowing you to use ANSI C code on LabVIEW Real-Time (RT) targets. This document discusses the details and benefits of using LabWindows/CVI to incorporate ANSI C code in LabVIEW Real-Time applications.

This document is intended for LabVIEW developers who want to include C or LabWindows/CVI DLLs in their projects.

NOTE – If you are a LabWindows/CVI developer, you can use the LabWindows/CVI Real-Time Module to download a DLL onto an RT target. Refer to *Programmer Reference»Creating and Downloading DLLs to Real-Time Targets* in the *LabWindows/CVI Help* contents for more information and a step-by-step guide for using LabWindows/CVI and RT targets. To access the *LabWindows/CVI Help*, select **Help»Contents**.

Table of Contents

Benefits of Using LabWindows/CVI DLLs in LabVIEW Real-Time

LabWindows/CVI can compile code into a LabVIEW Real-Time-compatible DLL that the LabVIEW Real-Time environment can call and execute. This feature reduces development time for real-time applications in the following ways:

- You can reuse large amounts of existing ANSI C code, which greatly reduces development time for LabVIEW Real-Time applications.
- You can develop portions of your LabVIEW Real-Time applications in ANSI C.
- You can take advantage of the LabWindows/CVI development environment to create LabVIEW Real-Time VISA drivers for non-NI PXI/CompactPCI hardware. Therefore, you can incorporate third-party hardware into LabVIEW Real-Time applications. For more information about this topic, refer to the document in the *See Also* section.

Architecture of LabWindows/CVI DLLs Running on LabVIEW Real-Time

When you deploy LabWindows/CVI DLLs to LabVIEW Real-Time, you must carefully design the LabWindows/CVI DLL architecture to ensure determinism and reliability. You must avoid creating full-scale programs in LabWindows/CVI and then simply porting the entire application to LabVIEW Real-Time by way of a DLL. Instead, you should deploy DLLs that are modular stand-alone tasks to LabVIEW Real-Time. Deploying a C-based analysis library to LabVIEW Real-Time is an example of such a task. Implementations of instrument and third-party hardware drivers are also modular components that promote determinism and reliability on LabVIEW Real-Time.

See Also:
Porting a Windows PCI Device Driver to LabVIEW Real-Time

Overview of LabVIEW Real-Time

It is difficult to implement highly reliable and deterministic (completing operations within a fixed amount of time) applications using general-purpose OSs such as Microsoft Windows and Mac OS. Reliable and deterministic applications require more control over all operations running on the system – a level of control that general-purpose OSs typically do not provide. Therefore, test and control engineers often turn to real-time OSs to implement applications that require deterministic performance or high reliability.

The preemptive scheduling capabilities of real-time OSs deliver a framework for deterministic performance. With this architecture, critical processes are guaranteed to receive time from the processor whenever needed. In addition, the added control over all processes running on the system delivers a higher degree of reliability than general-purpose OSs provide. With a real-time OS, the execution of critical test or control processes is separated from the user interface operations. This ensures that a crash on the host user interface system will not interfere with the execution of the critical real-time system.

The LabVIEW Real-Time Module extends the ease of use of LabVIEW graphical programming to generate reliable deterministic test and control systems. With LabVIEW and the LabVIEW Real-Time Module, you develop code on a host computer and download it to run embedded on a real-time hardware target. LabVIEW Real-Time applications run on devices including embedded PXI controllers, networked Compact FieldPoint modules, and compact vision systems. In addition to a development environment, the host computer provides the user interface for running systems.

LabWindows/CVI Functions Compatible with LabVIEW Real-Time

LabVIEW Real-Time hardware devices include an embedded real-time OS. As discussed in the *Overview of LabVIEW Real-Time* section above, the real-time OS is different from traditional OSs, such as Windows, and offers a slightly different set of functions. When you create a DLL in LabWindows/CVI, you can specify to include LabVIEW Real-Time compatibility. When you do so, LabWindows/CVI includes only the built-in LabWindows/CVI library functions that are compatible with the real-time OS.

Functions from the following native LabWindows/CVI libraries are compatible with the LabVIEW Real-Time Support Engine.

- Analysis or Advanced Analysis Library
-

ANSI C
- Formatting and I/O
- TCP Support Library
- Utility Library

In addition to the libraries listed above, you also can link the following libraries into the project.

- VISA Library
- NI-CAN Library
- NI-DAQmx Library
- NI-DMM
- NI-Scope
- NI-FGEN
- NI-Switch
- NI-HSDIO
- Most of the libraries in the `instrsup.dll`

NOTE – Not all of the functions in the preceding libraries are compatible. For a complete list of these exported functions, refer to *Programmer Reference»Creating and Downloading DLLs to Real-Time Targets»Using LabWindows/CVI Libraries in Real-Time DLLs* in the *LabWindows/CVI Help* contents.

TCP Support Library

LabWindows/CVI 7.0 and later includes TCP function support for LabVIEW Real-Time application development. With this feature, DLLs on real-time targets can share data directly with nodes on the network, which eliminates the need to return to the LabVIEW code to pass data to another node. TCP function calls, such as the write and read functions, trigger events when they return. In previous versions of LabWindows/CVI, these events were captured by messaging, which is not available under real-time OSs. However, the TCP Support Library `ProcessTCPEvents` function uses polling to capture TCP events, which is supported on real-time OSs.

This feature is useful in any application that must publish data as it is received. For example, you might have a DLL that contains a function with multiple data acquisition loops. You might want to publish the data as you acquire it. Because DLLs pass data back to the calling application only when the function has finished executing, you would need to be able to send the data from within the DLL. Without TCP polling functions, the only way to send the data across the network as you received it would be to implement a loop in LabVIEW and call a function in the LabWindows/CVI DLL that acquired data only a single time. Using this method, each time the data was acquired, the function would return and the data would be passed back to the LabVIEW Real-Time application. Once the data was back in LabVIEW, you could use the LabVIEW TCP functions to publish the data. This would require the DLL to be called, loaded, and unloaded each time the LabVIEW loop was executed, which greatly impacts performance. With the added functionality of the LabWindows/CVI TCP Support Library function calls, you can send the data from within the DLL in each loop. You have to call the DLL only once, which saves time and memory.

Debugging LabWindows/CVI DLLs on LabVIEW Real-Time Targets

LabWindows/CVI 7.1 introduced remote debugging, which allows you to debug DLLs while they are running on real-time targets. The debugger, which is on the local computer, and the debuggee, which is on the target computer, communicate over a TCP/IP connection.

Without this technology, DLLs specified for use on LabVIEW Real-Time targets would have to be debugged on the host machine. While you can detect and fix many errors using this method, there is no way to guarantee that the DLL would work when ported over to real-time targets because of the inherent differences between the Windows and real-time OSs. With remote debugging, you can use all of the debugging tools available on the host machine, such as the ability to set regular and conditional breakpoints, set the next statement, and view and change the value of variables, to ensure the DLL functions properly on LabVIEW Real-Time targets.

LabWindows/CVI Host Applications for LabVIEW Real-Time Targets

You can use LabWindows/CVI to create host applications for LabVIEW Real-Time targets. In doing so, you can use LabWindows/CVI applications to communicate with and control LabVIEW Real-Time targets. A practical example of this is a real-time target that acquires data and sends it to a host LabWindows/CVI application that then displays the data. The most common methods of communicating between the host application and the real-time target are TCP and UDP.

NOTE - When you use TCP, it is important to remember that LabVIEW uses Big Endian notation. If the program that you are sending the data to uses Little Endian notation, you must convert the data before you display the data. You can convert the data in LabVIEW using the Swap Bytes and Swap Words functions.

Application Example – Creating LabVIEW Real-Time DLLs in LabWindows/CVI

Objective: To convert an existing LabWindows/CVI program into a DLL that LabVIEW Real-Time targets can use.

Software Requirements

- LabWindows/CVI 7.0 and later
- `Calc.zip` file (This file is available from the LabWindows/CVI Standard I/O Calculator example on NI Developer Zone. A link to this example program is available at the end of this section.)

Instructions

1. Open `simpCalc.prj`.
2. Open `simpCalc.c`. Press <F2> to jump to the first tag. The four functions that you will export to the DLL are prototyped here. Press <F2> again to jump to the section of code in which each function is defined.
3. Compile and run the project. The program takes two numerical inputs, performs the operation the user specifies, and outputs the result.
4. Configure the project to generate a DLL by specifying Dynamic Link Library as the target type for the project. Select **Build»Target Type»Dynamic Link Library** as shown in Figure 1.

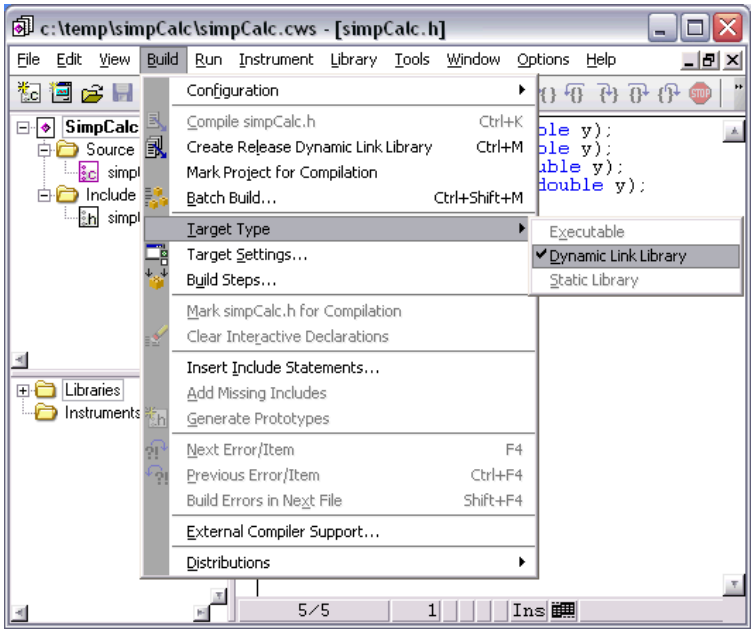

Figure 1. Specifying the Target Type

5. To specify to build the release version of the DLL, select **Build»Configuration»Release**.
6. LabWindows/CVI creates a DLL that exports only the functions that you specify. To export a function in the DLL, you must include the corresponding prototype in a header file. To create a header file, select **File»New»Include (*.h)**.
7. In the new header file, prototype the four functions that will be exported from the DLL. You can cut and paste the prototypes from `simpCalc.c`.
8. Save the header file as `simpCalc.h` and add the file to the project. The code in the header file should match the code in Figure 2.

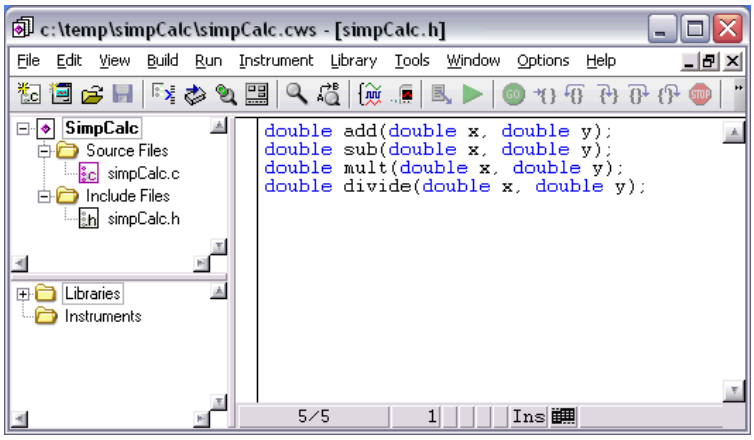

Figure 2. simpCalc Header File

9. Next, you will configure the target settings for the DLL that you will create. Select **Build»Target Settings** to open the Target Settings dialog box.
10. Specify the name and location of the DLL in the **DLL file** option of the Target Settings dialog box. By default, LabWindows/CVI sets the DLL name to that of the current project, in this example `simpCalc.dll`.
11. Select **Real-time only** in the **Run-time support** option. If you select **Real-time only**, LabWindows/CVI links to a smaller set of LabWindows/CVI built-in library functions that are supported in real-time applications.
12. Click **Change** to open the DLL Export Options dialog box. Select the `simpCalc.h` header file to ensure that only the functions that were prototyped in the header file will be exported from the DLL. When you finish, the Target Settings dialog box should look like Figure 3.

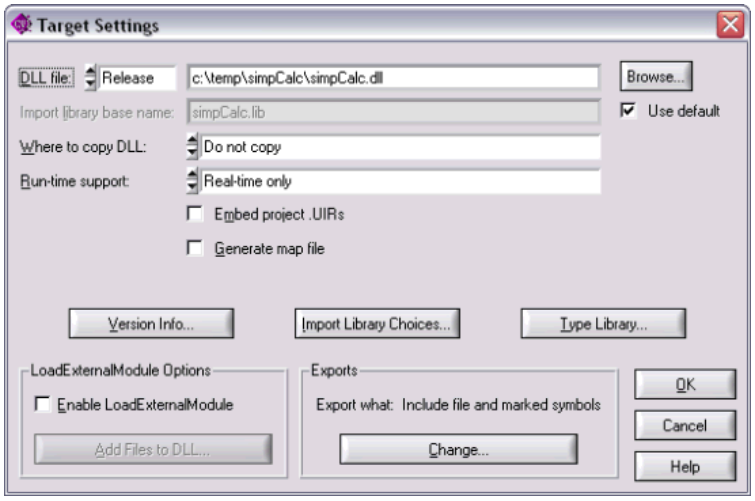

Figure 3. Target Settings Dialog Box

13. Click **OK** to close the dialog box.
14. Select **Build»Create Release Dynamic Link Library** to create the DLL.
15. LabWindows/CVI creates the DLL files and displays a message that lists the files created. You now are ready to call the LabWindows/CVI DLL from a LabVIEW Real-Time application.

See Also:
LabWindows/CVI Standard I/O Calculator

Application Example – Calling a LabWindows/CVI DLL from a LabVIEW Application

Objective: To call a LabWindows/CVI-created DLL from a LabVIEW application.

Software Requirements

- LabVIEW
- `simpCalc.dll` file (created in previous exercise)

Instructions

1. Open LabVIEW.
2. Refer to the *LabVIEW Help* for information about creating a real-time project.
3. Select **File»New VI** to create a new VI.
4. On the front panel, place two numeric controls and one numeric indicator. Label the controls **X** and **Y**. Label the indicator **Result**.
5. Call the LabWindows/CVI DLL that you created in the previous exercise. Place a Call Library Function Node onto the block diagram. The Call Library Function Node is located on the **Connectivity»Libraries & Executables** subpalette.
6. Double-click the Call Library Function Node. In the Call Library Function dialog box, click **Browse** to specify a DLL. Navigate to and select the **simpCalc.dll** file that you created in the previous exercise.
7. Select one of the four functions from the simpCalc project in the **Function Name** control.

   Note – Only the functions that were prototyped in the header file are available.
8. Specify the argument types for the return value and parameters. The simpCalc program takes two doubles as parameters and returns a double. The corresponding variable in LabVIEW is the numeric 8-byte double. For the first parameter, **return type**, specify **Numeric** for the **Type** and **8-byte Double** for the **Data Type**.
9. To specify the argument types for the parameters that are passed to the function, click **Add a Parameter After**. Specify **arg1** as a numeric 8-byte double. Repeat this process for **arg2**.
10. To take further advantage of LabVIEW Real-Time capabilities, you can specify the function to be reentrant. If a function is reentrant, multiple threads can call the function in the DLL simultaneously. In the **Run in UI Thread** drop-down box, select **Reentrant**. The Call Library Function Node on the front panel changes color to yellow to indicate the function is reentrant.
11. When you finish, the Call Library Function dialog box should look similar to Figure 4.

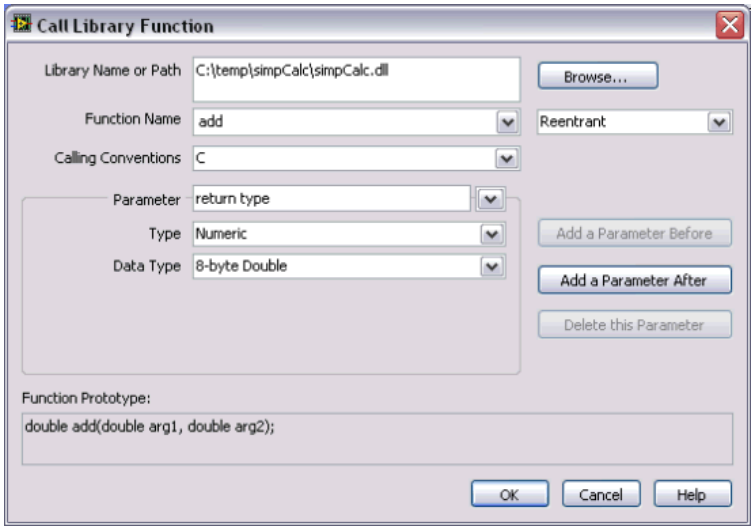

Figure 4. Call Library Function Dialog Box

12. Click **OK** to close the dialog box and return to the block diagram.
13. Notice that the Call Library Function Node now reflects the parameters that you specified. The Call Library Function Node has two input terminals and three output terminals. The top right terminal will return the result. The other two outputs are the two inputs passed through. Wire **X** to the top input and **Y** to the lower input. Wire the top output terminal to the **Result** indicator.
14. Save the program as `DllCaller.vi`. When you finish, the block diagram should look like Figure 5.

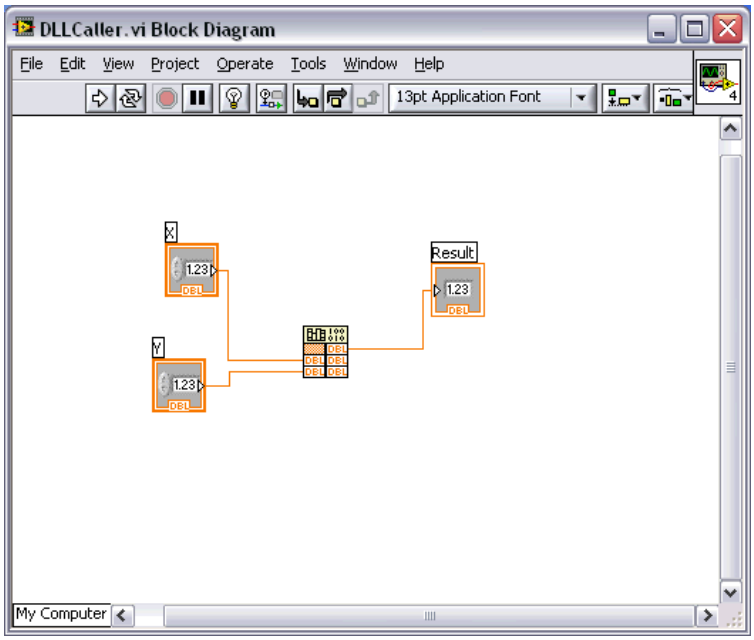

Figure 5. DllCaller.vi Block Diagram

15. You now are ready to test the program. Switch to the front panel and enter values for the **X** and **Y** inputs. Click the **Run** button. The numeric indicator displays the corresponding result.

Application Example – Creating a Debuggable LabVIEW Real-Time DLL in LabWindows/CVI

Objective: To convert an existing LabWindows/CVI program into a debuggable DLL that can be called and debugged on a LabVIEW Real-Time target.

Software Requirements

- LabWindows/CVI 7.1 or later
- `Calc.zip` file (This file is available from the LabWindows/CVI Standard I/O Calculator example on NI Developer Zone. A link to this example program is available at the end of this section.)

Instructions

1. Open `DebCalc.prj`.
2. Configure the project to generate a DLL by specifying Dynamic Link Library as the target type for the project. Select **Build»Target Type»Dynamic Link Library**.
3. To specify the DLL to be debuggable, select **Build»Configuration»Debug**.
4. LabWindows/CVI creates a DLL that exports only the functions that you specify. To export a function in the DLL, you must include the corresponding prototype in a header file. To create a header file, select **File»New»Include (*.h)**.
5. In the new header file, prototype the four functions that will be exported from the DLL. You can cut and paste the prototypes from `DebCalc.c`.
6. Save the header file as `DebCalc.h` and add the file to the project. The code in the header file should match the code in Figure 6.

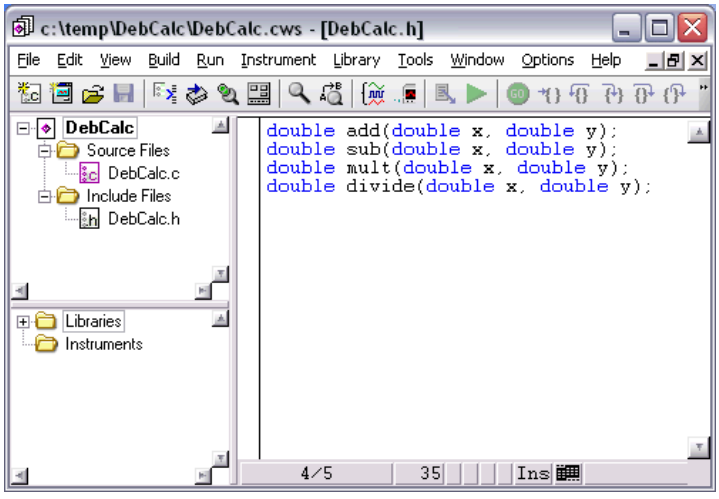

Figure 6. DebCalc Header File

7. Configure the target settings for the DLL that you will create. Select **Build»Target Settings** to open the Target Settings dialog box.
8. Specify the name and location of the DLL in the **DLL file** option of the Target Settings dialog box. For this example, the resulting DLL is named `DebCalc.dll`.
9. Select **Real-time only** in the **Run-time support** option. If you select **Real-time only**, LabWindows/CVI links to a smaller set of LabWindows/CVI built-in library functions that are supported in real-time applications.
10. Click the **Change** button to open the DLL Export Options dialog box. Select the `simpCalc.h` header file to ensure that only the functions that were prototyped in the header file will be

exported from the DLL. When you finish, the Target Settings dialog box should look like Figure 7.

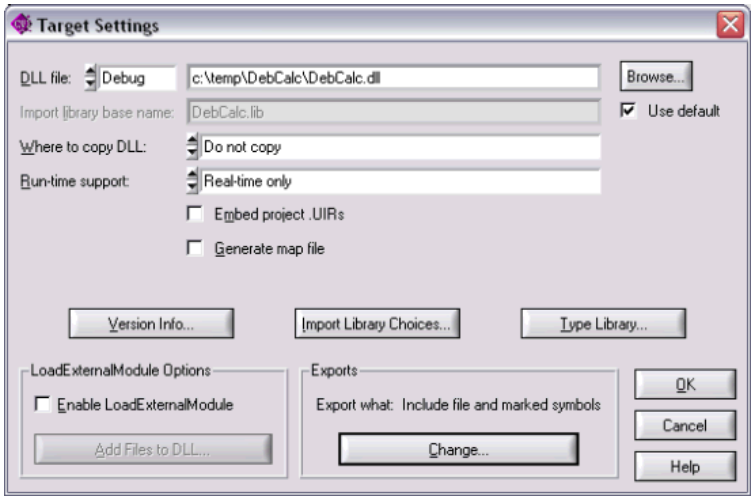

Figure 7. Target Settings Dialog Box

11. Click **OK** to close the dialog box.
12. Select **Run»Switch Execution Targets** to view a list of available RT targets. Select the RT target you want to use from the list. To configure a new RT target, select **Select Target with Options**.
13. In the Select Target with Options dialog box, select **New RT Target via LabVIEW on the Network** from the ring control at the top of the dialog box.
14. Enter the machine name or IP address of the real-time target in the **Machine Name/IP** option. Enter the network port for communication between LabWindows/CVI and the real-time target in the **Port** option. Click **OK** to exit the dialog box.
15. Select **Build»Create Debuggable Dynamic Link Library** to create the DLL.
16. LabWindows/CVI creates the DLL files and displays a message listing the created files. You now are ready to call the LabWindows/CVI debuggable DLL from a LabVIEW Real-Time application.

See Also:
LabWindows/CVI Standard I/O Calculator

Application Example – Debugging LabWindows/CVI DLLs on LabVIEW Real-Time Targets

Objective: To become familiar with the LabWindows/CVI remote debugging utilities and to debug a DLL that is running on a LabVIEW Real-Time target.

Software Requirements

LabWindows/CVI 7.1 and later
LabVIEW Real-Time
`DebCalc.dll` (created in previous exercise)
`Calc.zip` file (This file is available from the LabWindows/CVI Standard I/O Calculator example on NI Developer Zone. A link to this example program is available at the end of this section.)

Instructions

1. Open LabVIEW.
2. Refer to the LabVIEW Help for information about creating a real-time project.
3. Select **File»New VI** to create a new VI.
4. On the front panel, place two numeric controls and one numeric indicator. Label the controls **X** and **Y**. Label the indicator **Result**.
5. Call the LabWindows/CVI DLL that you created in the previous exercise by placing a Call Library Function Node onto the block diagram. The Call Library Function Node is located on the **Connectivity»Libraries & Executables** subpalette.
6. Double-click the Call Library Function Node. In the Call Library Function dialog box, click **Browse** to specify a DLL. Navigate to and select the **DebCalc.dll** file that you created in the previous exercise.
7. Select one of the four functions from the DebCalc project in the **Function Name** option.

   Note – Only the functions that were prototyped in the header file are available.
8. Specify the argument types for the return value and parameters. For the first parameter, **return type**, specify **Numeric** for the **Type** and **8-byte Double** for the **Data Type**.
9. To specify the argument types for the parameters that are passed to the function, click **Add a Parameter After**. Specify **arg1** as a **Numeric 8-byte Double**. Repeat this process for **arg2**.
10. To take further advantage of LabVIEW Real-Time capabilities, you can specify the function to be reentrant. If a function is reentrant, multiple threads can call the function in the DLL simultaneously. In the **Run in UI Thread** drop-down box, select **Reentrant**. The Call Library Function Node on the front panel changes color to yellow to indicate the function is reentrant.
11. When you finish, the Call Library Function dialog box should look similar to Figure 8.

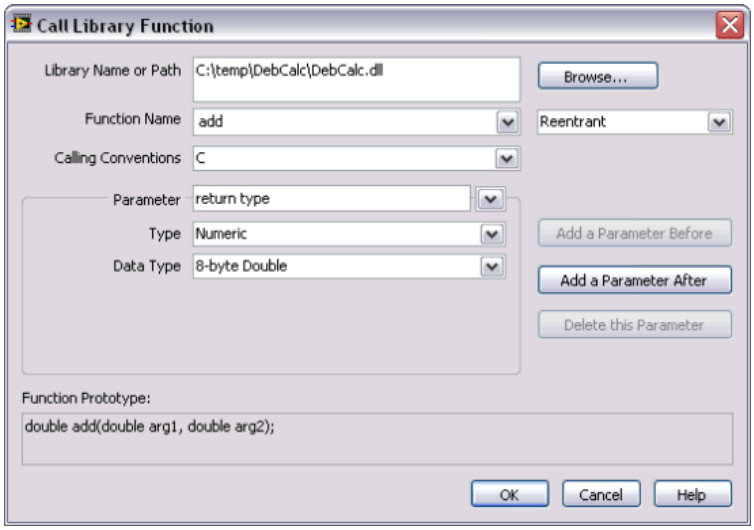

Figure 8. Call Library Function Dialog Box

12. Click **OK** to close the dialog box and return to the block diagram.
13. Notice that the Call Library Function Node icon now reflects the parameters that were specified. The Call Library Node has two input terminals and three output terminals. The top right terminal will output the result. The other two outputs are the two inputs passed through. Wire **X** to the top input and **Y** to the lower input. Wire the top output terminal to the **Result** indicator.
14. Save the program as `DebugDllCaller.vi`. When you finish, the block diagram should look like Figure 9.

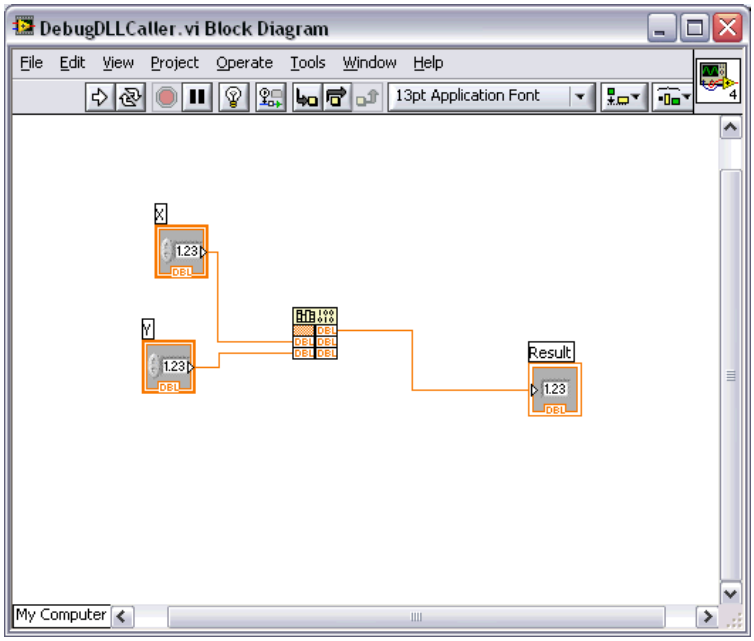

Figure 9. DebugDLLCaller.vi Block Diagram

15. You now are ready to run the program and debug function calls to `DebCalc.dll`. Switch to the front panel and enter values for the **X** and **Y** inputs.
16. In LabWindows/CVI, open `DebCalc.prj`.
17. Select **Run»Debug Project** to start the debugger. The debugger acts as a TCP server and waits for the DLL to connect. After you start the debugger, LabWindows/CVI displays a message that instructs you to download the DLL onto the LabVIEW Real-Time hardware.
18. Return to the VI you created in LabVIEW and select **Operate»Run**.
19. When the DLL function finishes executing, LabWindows/CVI displays a message box indicating that the debugger is waiting for additional connection requests from the DLL. To continue debugging the DLL, ignore the message and run the LabVIEW application again. If you are finished debugging, click the **Stop Debugging** button in this message box.

Conclusion

By using LabWindows/CVI to create and debug LabVIEW Real-Time-compatible DLLs, you can extend the functionality of LabVIEW Real-Time. You can use these DLLs to incorporate ANSI C code into deterministic, reliable real-time applications. The application examples in this document have demonstrated how to create both release and debug versions of LabVIEW Real-Time-compatible DLLs and how to call these DLLs from within a LabVIEW application. You can use the steps these examples describe to create your own LabVIEW Real-Time-compatible DLLs in LabWindows/CVI.

See Also:
LabWindows/CVI Standard I/O Calculator