FAQ: Using Dynamic Link Libraries with NI LabWindows™/CVI™

Publish Date: May 19, 2014

Overview

This document answers frequently asked questions about creating and using DLLs in LabWindows/CVI.

Table of Contents

1. How Do I Create DLLs with LabWindows/CVI?

You can use LabWindows/CVI to create 32-bit DLLs. When you use LabWindows/CVI to create a DLL, LabWindows/CVI also creates an import library that you can include in projects that use the DLL. This is referred to as statically linking the DLL.
You must create a separate project for each DLL you want to generate. Complete the following steps to create a DLL using LabWindows/CVI:

1. Select **Build >> Target Type >> Dynamic Link Library** to specify the target settings for the DLL project.
2. Use the **Build >> Configuration** submenu to specify whether to build a release or debuggable version of the DLL.
3. Once you have the source code for the DLL ready (see the How Do I Prepare My C Code for Use in a LabWindows/CVI DLL? section of this FAQ for information), select **Build >> Create Dynamic Link library** to build the DLL.

If you are creating a LabWindows/CVI DLL for use with LabVIEW Real Time, you must specify that in the Target Settings dialog box. Select **Build >> Target Settings** and then select **Real-time only** in the **Run-time support** control.

For more information about the menu options discussed in this section, refer to the Creating DLLs in LabWindows/CVI topic in the LabWindows/CVI Help.

2. How Do I Prepare My C Code for Use in a LabWindows/CVI DLL?

When you create a DLL, you must address the following issues, which can affect your source code and header file. Refer to the Preparing Source Code for Use in a DLL topic in the LabWindows/CVI Help for more information about how to address these issues.

Using the appropriate calling convention to declare the functions you want to export

If you intend for only C or C++ programs to use your DLL, you can use the __cdecl convention to declare the functions to export. However, if you want your DLL to be called from environments such as Microsoft Visual Basic, you must declare the functions to export with the __stdcall calling convention.

The following example demonstrates the use of __cdecl with a function:
int __cdecl MyCDeclFunction(const char *);

The following example demonstrates the use of __stdcall with a function:
int __stdcall MystdCallFunction(const char *);

If you do not specify a calling convention, LabWindows/CVI uses the **Default calling convention** specified under **Options >> Build Options**. For more information, refer to the Calling Convention for Exported Functions topic in the LabWindows/CVI Help. Nevertheless, National Instruments recommends that you always explicitly specify a calling convention when you create DLLs to be used in third-party situations. This ensures that clients that use your DLL automatically use the correct calling convention when they include the DLL header file in their project.

Specifying which DLL functions and variables you want to export

When a program uses a DLL, it can access only the functions or variables that the DLL exports. The DLL can export only globally declared functions and variables. The DLL cannot export functions and variables you declare as static.

Refer to the Exporting DLL Functions and Variables topic in the LabWindows/CVI Help for more information.

Marking imported symbols in the DLL header file you distribute

If your DLL might be used in a C or C++ environment, you must distribute a header file with your DLL. The clients that use your DLL will require a header file to get information about the functions that your DLL exports and the parameters and calling conventions for those functions. You can distribute the same header file you use for developing the DLL, if you appropriately mark the symbols for export. For more information, refer to the Marking Imported Symbols in an Include File Distributed with a DLL topic in the LabWindows/CVI Help.

The Microsoft Developers Network (MSDN) provides additional information about the calling conventions described here.

See Also:
MSDN: __stdcall calling convention
MSDN: __cdecl calling convention

3. What Are the Minimum Requirements of a LabWindows/CVI DLL?

To use a LabWindows/CVI DLL, you must have the LabWindows/CVI Run-Time Engine installed on the system. You also must make accessible any other files the DLL requires. For more information about the LabWindows/CVI Run-Time Engine, refer to the LabWindows/CVI Run-Time Engine topic in the LabWindows/CVI Help.

4. Are There Special Considerations for Creating LabWindows/CVI DLLs for Use with Other Development Environments?

You can use DLLs you create with LabWindows/CVI from any development language that supports the use of C-style DLLs. This includes development environments such as LabVIEW, TestStand, Visual C++, and Visual Basic 6.0. Each development environment has different ways of calling the DLL and has different limitations. For example, Visual Basic 6.0 can only call DLLs that have their calling convention defined as __stdcall. For information about using a LabWindows/CVI C DLL in one of these environments, refer to the documentation for the development environment. The Recommendations for Creating a DLL topic in the LabWindows/CVI Help provides a list of recommendations you can follow to ensure your LabWindows/CVI DLL works with most development environments.

It is easier to use a DLL in some development environments, such as Visual Basic 6.0 and TestStand, if you include a type library with your DLL. The type library provides more detailed information about the programming interface used to access the DLL. To embed a type library in the DLL, you must have a function panel (*.fp) file for the DLL. (In LabWindows/CVI 7.0 and later, you can select **Options >> Generate Function Tree** to generate a function panel file from a header file.) Once you create a function panel file, you can specify that this function panel be used to embed a type library in the DLL. Select **Build >> Target Settings** to open the Target Settings dialog box, click the **Type Library** button, enable the **Add type library resource to DLL** option, and specify the function panel file you created in the **Function panel file** control.

5. How Can I Call Third-Party DLLs from LabWindows/CVI?

LabWindows/CVI executables and DLLs can load 32-bit DLLs. To call a DLL from LabWindows/CVI, you must have the DLL header .h file and the import library .lib file. If you do not have the DLL import library, you can create one using the tools LabWindows/CVI provides. For more information, refer to the What Is the Difference between Static Linking to a DLL and Dynamic DLL Linking? section of this FAQ.

Because LabWindows/CVI is an ANSI C compiler, it can call only functions that have been exported as C-style functions. For more information, refer to the How Can I Use DLLs Created in Visual C++ with LabWindows/CVI? section of this FAQ.

6. What Is the Difference between Static Linking to a DLL and Dynamic DLL Linking?

LabWindows/CVI allows linking to DLLs using the 32-bit DLL import libraries that you generate when you create 32-bit DLLs. You can link an import library to your application in several ways, the most common of which is to add the DLL import library .lib to the LabWindows/CVI project that uses the DLL. If you add the DLL import library to the project, the project is statically linked or linked implicitly to the DLL. This means that the application the project generates automatically loads the linked DLL when the executable/DLL is loaded. Windows must be able to locate the linked DLL when it loads the executable/DLL, or it generates an error and shuts down the application.

In some cases, you might not have an import library or you might want to load the DLL at run time because you want to use it conditionally, based on some logic in your program. You also might want to first make sure that the DLL exists before the program loads it so that the application does not shut down if the DLL is missing. In these cases, you can dynamically load the DLL or link explicitly to it by using the Windows SDK LoadLibrary and GetProcAddress functions. Refer to National Instruments KB 2EIBT1Y1 - How Can I Access DLL Functions in a LabWindows/CVI Program without Including the Import Library in the Project, which is linked in the See Also section, for more information about how to use these functions in LabWindows/CVI.

Hint: If LoadLibrary is unable to find the DLL, it displays an error dialog box. Use the Windows SDK SetErrorMode function to prevent LoadLibrary from displaying the error dialog box.

See Also:
MSDN: Implicitly Linking to DLL
MSDN: Explicitly linking to DLL
National Instruments KB 2EIBT1Y1
MSDN Platform SDK: SetErrorMode

7. How Can I Create and/or Use Import Libraries in LabWindows/CVI?

When you use LabWindows/CVI to create DLL export functions or variables, LabWindows/CVI automatically generates an import library .lib for you. Depending on your compatibility mode, the import library is compatible with either Microsoft Visual C++ or Borland. These compilers use different import library formats, so you must generate different import libraries for them. You can generate import libraries for both compilers simultaneously by changing the default import library choices in the Target Settings dialog box. Refer to the Compiler/Linker Issues topic in the LabWindows/CVI Help for more information about using LabWindows/CVI with the supported external compilers.

LabWindows/CVI also can generate import libraries for third-party C-style DLLs, provided you have the header .h file for the DLL. For more information, refer to the Generating an Import Library topic in the LabWindows/CVI Help.

To use an import library from LabWindows/CVI, add the library to the project. Select **Edit >> Add Files to Project >> Library (*.lib)** to manually add the appropriate import library to the LabWindows/CVI project. Adding the import library ensures that the DLL for which the import library was generated is loaded into memory when the application is launched. If you want to load the DLL only when it is needed, you can use LoadExternalModule to load the import library and hence load the DLL and execute the code in the DLL. This ensures that the DLL is not loaded automatically when the application is launched and gives you finer control over DLL loading. For more information, refer to the LabWindows/CVI function help for LoadExternalModule.

When you use LoadExternalModule, you must distribute the import library along with the DLL for your application to work correctly. If you want to distribute only the DLL with your application and retain the ability to load the DLL explicitly, refer to the What Is the Difference between Static Linking to a DLL and Dynamic DLL Linking? section of this FAQ.

If you do not use LoadLibrary or LoadExternalModule and you do not add the import library to the project, you will get linker errors similar to the following error when building your application:

Undefined symbol 'DLLFunctionName@0' referenced in "file.c".

In this case, DLLFunctionName is the name of the function that you try to call in your application but is actually implemented in a DLL.
For more information about the ways you can use an import library with LabWindows/CVI, refer to the Rules for Using DLL Files topic in the LabWindows/CVI Help.

8. How Can I Use the Windows SDK from LabWindows/CVI?

You can use the Windows SDK functions in LabWindows/CVI. From the LabWindows/CVI Full Development System installation options, you can select to install the full Windows SDK. This option installs all the header files and libraries you need to call the Windows SDK functions. If you do not install the full Windows SDK or you have the LabWindows/CVI Base Package, you can use only a small subset of the Windows SDK functions.

To use a particular Windows SDK function, refer to the Microsoft Developers Network (MSDN) for the required import library .lib and required header .h file that must be included in the source. By default, LabWindows/CVI automatically loads some commonly used Windows SDK import libraries so you do not have to manually add them to the project. Refer to the Automatic Loading of SDK Import Libraries topic in the LabWindows/CVI Help for a list of these import libraries. If you use a function that is not included in any of these auto-loaded import libraries, select **Edit >> Add Files to Project >> Library (*.lib)** to manually add the appropriate import library to the LabWindows/CVI project.

See Also:
MSDN Home

9. How Can I Use DLLs Created in Visual C++ with LabWindows/CVI?

If you are creating a DLL using Visual C++ for use with LabWindows/CVI, you must mark the exported functions with extern "C" to specify that these functions will be used with a C compiler. If you do not mark an exported function with extern "C", LabWindows/CVI will not be able to call that function because C++ compilers mangle function names when they are exported from a DLL. The following code is an example of how the function might be declared in Visual C++ to specify C linkage and to specify that this function is to be exported.

```
extern "C" char __declspec(dllexport) GetChar( void )
```

For more information about extern "C", refer to the documentation for Visual C++.

Hint: If you believe that you are not exporting the correct function names from a DLL, you can use a free tool called Dependency Walker to view the function names that are exported. You can use this tool to view the exported function names and any dependencies your DLL might have. See the link in the See Also section to download this tool. The sign of a mangled function name is that it includes characters such as '@' and '?' in the name. For example, the function FPi_i@@YAHPAH@Z is a mangled version of the function FPi_i(int *a).

See Also:
MSDN: Linkage to Non-C++ Functions
Dependency Walker

10. Additional Windows DLL Resources

The Microsoft Developers Network (MSDN) provides information about DLLs on Windows systems. Refer to the following link to learn more about DLLs.

See Also:
MSDN: About Dynamic-Link Libraries

Related Links:
Exporting Variables and Functions from a DLL in LabWindows/CVI
How Can I Access DLL Functions in a LabWindows/CVI Program without Including the Import Library in the Project?
Creating a CVI DLL that Contains a UIR File and Using the DLL in Visual C++

Next Steps

See the Latest LabWindows/CVI Pricing Information
Evaluate LabWindows/CVI
LabWindows/CVI Home Page