

LabVIEW™ Embedded Development Module Release Notes

Version 2.5

Contents

Introduction	2
Overview of Porting LabVIEW to Any 32-Bit Microprocessor.....	2
Customizing the Build Process	2
Porting the Run-Time Library	3
Creating an I/O Interface	3
Implementing Target Syntax Checking	4
Prerequisites	4
System Requirements and Target Recommendations.....	5
Host Computer Requirements.....	5
Target Recommendations	5
Installation.....	6
LabVIEW Embedded Development Module Example Targets.....	6
Upgrading from Version 2.0	7
Changes in the LabVIEW Development System.....	7
Upgrading Target Support	7
Upgrading the Redboot Bootloader for the PHYTEC Example Target	7
Upgrading Instrumented Debugging	8
Logging	9
What's New in Version 2.5	10
New Freescale ColdFire M5329 Example Target	10
VI-Specific Code Generation Options	10
Expression Folding	11
New Embedded Development Module Functions Palette	11
Target Server.....	11
Updated Embedded Automatic Test Framework.....	12
Dynamically Calling VIs on a Target	12

Background Debugging Thread for Instrumented Debugging	13
Newly Supported VIs and Functions	14
Incremental Builds on Example Targets	14
Thread-Safe and Interrupt-Safe VIs	15
Thread Safe	15
Interrupt Safe	15
Embedded Development Module Documentation	15
Where to Go for Support	16

Introduction

The LabVIEW Embedded Development Module allows you to port LabVIEW to any 32-bit microprocessor for system deployment. By using a single development tool from concept to finished product, you can ease the development process and increase end quality while reducing time to market. For more information on the Graphical System Design process, visit ni.com/embedded.

Overview of Porting LabVIEW to Any 32-Bit Microprocessor

When you are ready to deploy your embedded design, you can use the LabVIEW Embedded Development Module along with a third-party toolchain and an embedded operating system to extend the graphical LabVIEW programming experience to any 32-bit microprocessor.

The following list highlights the main areas of the porting process.

- Customizing the build process
- Porting the run-time library
- Creating an I/O interface
- Implementing target syntax checking

Refer to the *LabVIEW Embedded Development Module Porting Guide*, available by selecting **Start»All Programs»National Instruments»LabVIEW 8.5»LabVIEW Manuals** and opening `EMB_Porting_Guide.pdf`, for detailed information about the porting process.

Customizing the Build Process

Customizing the build process involves editing the plug-in VIs that control C code generation, compilation, linking, and downloading. Each step of customizing the build process has a set of plug-in VIs that define the behavior.



Note Plug-in VIs give LabVIEW the functionality to interact with your embedded target through items such as shortcut menu commands and property configuration dialog boxes. Plug-in VIs also provide support for the toolchain and OS for your target.

The LabVIEW C Code Generator is part of the LabVIEW executable. You access the LabVIEW C Code Generator with a VI server call that contains the parameters controlling the code generation settings. The VI server call also includes such things as endianness, memory model, and guard code removal.

The compiling and linking VIs script your C compiler to do the work of building the executable. You can customize the build settings to add extra options to control the linker. For example, you can define the executable format the linker generates. The download VIs control the transfer of the executable from the desktop PC to the embedded device.

Refer to Chapter 3, *Using the Target Editor to Manage Embedded Targets*, in the *LabVIEW Embedded Development Module Porting Guide*, for more information about customizing the build process and creating plug-in VIs for your target.

Porting the Run-Time Library

The run-time library consists of code containing such things as communications, data manipulation, and timing functions. The run-time library is distributed in source code form. To port the run-time library to a new OS, you copy OS-specific folders from a similar existing OS to a new folder with the new OS name. You then modify these files to work on the new OS.

The main features that require porting are serial communications, TCP/IP, time functions, events, threads, critical sections, and `printf` output. You can port the rest of the run-time code by modifying the code in `LVDefs_plat.h` and `LVSysIncludes.h`, which contain macros and constants that might change when porting one OS to another.

Refer to Chapter 7, *Porting the LabVIEW Embedded Run-Time and Analysis Libraries Source Code*, in the *LabVIEW Embedded Development Module Porting Guide*, for more information about porting the run-time library for your target.

Creating an I/O Interface

Elemental I/O Nodes provide a mechanism for portable I/O. Use the Elemental I/O Device Wizard to define the I/O characteristics of a board and VIs that govern the behavior of the various types of Elemental I/O Nodes, such as analog input, analog output, digital input, digital output,

digital bank input, digital bank output, and pulse width modulation output nodes.

For each type of I/O, you create one or more VIs that perform the I/O operation and one or more VIs that configure the operation. For example, you can implement the I/O VIs using an Inline C Node to perform I/O register operations. The configuration VIs provide the interface for a user to configure parameters, such as channels, that can vary from one node to another.

Refer to Chapter 9, *Elemental I/O*, in the *LabVIEW Embedded Development Module Porting Guide*, for more information about creating I/O for your target.

Implementing Target Syntax Checking

Target syntax determines the supported and unsupported features for a specific target. You receive a broken **Run** button when you use an unsupported feature for a target. Target syntax checking is useful when importing VIs that might contain unsupported features. LabVIEW loads a configuration file when you switch to a different application instance that contains the target syntax checking for that target. Refer to the *Working with Application Instances* topic in the *LabVIEW Help* for more information about application instances.

You can customize the LabVIEW palettes and target syntax checking for each target so users can avoid using features the target does not support. For example, you can remove the TCP/IP VI and functions from the palettes and list of supported VIs and functions for targets without Ethernet connectivity.

Refer to Chapter 18, *Configuring the Target Syntax*, in the *LabVIEW Embedded Development Module Porting Guide*, for more information about implementing target syntax checking for your target. Refer to Chapter 3, *Creating Icons and Palettes*, in the *LabVIEW Embedded Development Module Target Distribution Guide*, for more information about customizing palettes.

Prerequisites

Before you can begin porting LabVIEW to a new target, you must be able to build, download, and run a “hello world” application in C, which verifies that the necessary toolchain is installed on the host computer and the board support package (BSP) is correctly installed and configured.

To use the Embedded Development Module to port LabVIEW to a new target, you should be knowledgeable about the following:

- LabVIEW
- C
- C compiler toolchain for your target
- Debugging an application on your target outside of LabVIEW
- Your target

System Requirements and Target Recommendations

Host Computer Requirements

The Embedded Development Module has the following requirements:

- A computer with Windows Vista/XP/2000
- LabVIEW 8.5 Full or Professional Edition
- NI-VISA Run-Time Engine version 3.1 or later (available for download at ni.com)

Refer to the *LabVIEW Release Notes*, available by selecting **Start» All Programs»National Instruments»LabVIEW 8.5»LabVIEW Manuals** and opening *LV_Release_Notes.pdf*, for standard LabVIEW development system requirements.

Target Recommendations

National Instruments recommends your target has the following:

- 32-bit processor architecture.
- 256 KB of application memory (in addition to the embedded operating system memory requirements).
- An embedded operating system so you can take advantage of the parallelism and multithreading in LabVIEW.
- A hardware floating-point unit or floating-point emulation library to perform the math calculations in the LabVIEW analysis library, which is floating-point. You can improve performance by performing the math calculations in hardware instead of using software emulation.

Installation

Complete the following steps to install the Embedded Development Module.

1. Log on as an administrator or as a user with administrator privileges.
2. Install LabVIEW 8.5, if not already installed.
3. Install the Embedded Development Module 2.5.
4. Activate the Embedded Development Module.

You must activate the Embedded Development Module to access the example targets. You have the option of activating the Embedded Development Module at the end of installation. You also can use the NI License Manager, available by selecting **Start»All Programs»National Instruments»NI License Manager**, to activate National Instruments products. Refer to the *National Instruments License Manager Help*, available by selecting **Help»Contents** in the National Instruments License Manager, for more information about activating NI products.



Tip If you have beta versions, previous versions, and/or multiple versions of the Embedded Development Module and the activation is not successful at the end of installation, use the NI License Manager to activate the software.

5. Restart the computer when the installer completes.

LabVIEW Embedded Development Module Example Targets

The Embedded Development Module includes several example targets. You can use the example targets as a starting point when you create new embedded targets. Refer to the *LabVIEW Embedded Development Module Porting Guide* for information about how to set up the example targets and which features are implemented on which example target.

None of the example targets are meant to be fully featured, ready-to-use targets. The different example targets have different implementations to show you various ways of implementing a feature. When you are implementing a feature for a new embedded target, look for an existing implementation in an existing target that might be similar to your target.



Note You must activate the Embedded Development Module to access the example targets.

Upgrading from Version 2.0

Changes in the LabVIEW Development System

Refer to the *LabVIEW 8.5 Upgrade Notes* or the *LabVIEW 8.5 Features and Changes* topic in the *LabVIEW Help* for information about new features and changes.

Upgrading Target Support

Because each target implementation is different, upgrading target support for version 8.5 of LabVIEW is different for every target. In general, you must complete the following steps to upgrade your target support.

1. Create a folder in `LabVIEW 8.5\Targets`. National Instruments recommends using some version of your company name.
2. Create an `Embedded` folder under the `LabVIEW 8.5\Targets\<company name>` directory.



Note LabVIEW does not recognize any targets outside of this directory.

3. Copy your target directory from LabVIEW 8.2, which is located in `LabVIEW 8.2\Targets\<company name>`, to the new LabVIEW 8.5 directory you created in steps 1 and 2.
4. Select **Tools»Embedded Tools»Target Editor** in LabVIEW 8.5 to launch the Target Editor to modify the `TgtSupp.xml` file. LabVIEW uses the `TgtSupp.xml` file to incorporate your target into LabVIEW. Refer to the *LabVIEW Embedded Development Module Porting Guide*, available by selecting **Start»All Programs»National Instruments»LabVIEW 8.5»LabVIEW Manuals**, for more information about using the Target Editor to manage embedded targets.
5. Open all of your existing plug-in VIs in LabVIEW 8.5 and verify none of the VIs are broken.
6. Move any target-specific VIs and palettes from the LabVIEW 8.2 directory to the LabVIEW 8.5 directory.

Upgrading the Redboot Bootloader for the PHYTEC Example Target

You must upgrade the Redboot bootloader on the PHYTEC example target to use the target in version 2.5 of the Embedded Development Module.



Tip Disconnect the GPIO extension board before programming the PHYTEC board.

Refer to the *LabVIEW Embedded Development Module Porting Guide* for detailed instructions on installing the RedBoot bootloader.



Note The RedBoot bootloader is not backward compatible. If you want to use the PHYTEC example target with version 2.0 of the Embedded Development Module, you must reinstall the RedBoot bootloader included in version 2.0.

Upgrading Instrumented Debugging

Version 2.5 adds two new C functions and add two new function pointers. In addition, the `PDAXxxDebugConnect` and `PDAXxxDebugDisconnect` functions have changed behavior.

If You Use TCP/IP for Instrumented Debugging

You do not have to implement the new functions and function pointers if you are using TCP/IP for debugging because they have already been implemented.

In version 2.0 of the Embedded Development Module, you call the `nitargetStartDebug` VI and then run the embedded application.

In version 2.5, the plug-in VI you call from the **Debug** item in the shortcut menu must run the embedded application and then call the `nitargetStartDebug` VI.

If You Use a Protocol Other than TCP/IP for Instrumented Debugging

If you use a protocol other than TCP/IP for instrumented debugging, you must implement the two new C functions and add two new function pointers.

Implementing the Communication Layer for Your Target

You must create an instance of two new C functions in the LabVIEW Embedded Run-Time Library. Refer to the *Implementing the Instrumented Debugging Communication Layer for Your Embedded Target* section in the *Instrumented Debugging* chapter of the *LabVIEW Embedded Development Module Porting Guide* for more information about implementing instrumented debugging and the implementations in the `labview\CCodeGen\libsrc\comms` directory. In addition to these two new functions, you must still implement the `Connect`, `Disconnect`, `Write`, `Read`, and `Bytes Available` functions. Refer to the [Background Debugging Thread for Instrumented Debugging](#) section in this document for more information about the new background debugging thread.

The following are the two new C functions:

- Prepare Connection

```
eRunStatus PrepareConnection(void)
```

Prepares a debugging connection by opening the port in non-blocking mode. Returns `eFail` for an error. Otherwise, it returns `eFinished`.

- Release Connection

```
eRunStatus ReleaseConnection(void)
```

Releases the resources allocated by the prepare connection function. Returns `eFail` for an error. Otherwise, it returns `eFinished`.

Adding Function Pointers

You must add the following new function pointers in

```
PDADebugInitializeComm ():
```

- `PDADebugPrepareConnection`—Assign this pointer to the `PrepareConnection` function.
- `PDADebugReleaseConnection`—Assign this pointer to the `ReleaseConnection` function.

These pointers are in addition to the existing function pointers. Refer to the *Assigning to Function Pointers* section in the *Instrumented Debugging* chapter of the *LabVIEW Embedded Development Module Porting Guide* for more information about the existing function pointers.

Behavior Change in Existing Functions

In previous versions, the following two functions are blocking until the host computer connects. These functions are now non-blocking.

- `PDAXXXDebugConnect` must connect right away or return `eNotFinished`. This function is periodically called until the connection is successful. The listening socket must be non-blocking.
- `PDAXXXDebugDisconnect` now closes the current connection socket. This function does not close the listening socket.

Logging

LabVIEW logs all of its actions, such as loading plug-in VIs, if you create a log file, `Embedded.log`, and place the file in the `labview` directory before launching LabVIEW. This log file is useful for debugging porting because you can quickly locate the plug-in VIs causing errors. You cannot specify which actions LabVIEW logs.



Tip Use Notepad or some other text editor to create an empty text file, and then rename the file `Embedded.log`.

What's New in Version 2.5

Version 2.5 of the Embedded Development Module includes the following new features:

New Freescale ColdFire M5329 Example Target

The Embedded Development Module includes a uClinux image that works with M5329EVB and M5329EVE evaluation boards. The new ColdFire example target includes a pre-built run-time library and an example implementation of TCP-based instrumented debugging.

You must configure the board before you can use it as a target in LabVIEW. Refer to the *LabVIEW Embedded Development Module Porting Guide*, available by selecting **Start»All Programs»National Instruments»LabVIEW 8.5»LabVIEW Manuals**, for information about configuring the board to use with LabVIEW.

VI-Specific Code Generation Options

You now can set code generations options on a per VI basis. In previous versions of the Embedded Development Module, you can set the code generation options only on a per project level. Configuring code generation options per VI gives you more control over how the LabVIEW C Code Generator generates C code.

To set code generation options for a VI, select **File»VI Properties** from the front panel window or block diagram window. Select **C Code Generation Options** from the **Category** pull-down menu. You can select **Use project setting, True, or False**. Select **True** to enable the option and **False** to disable it. The default is **Use project setting**, so you only need to set the options in **VI Properties** if you want to override the code generation settings in the project.

The **Build Specification Properties** dialog box includes a new **Source File Settings** category. Use this page to see which VIs have which C code generation options. If you use the **VI Properties** dialog box to set a code generation option for a VI, you cannot change the option in the **Build Specification Properties** dialog box. This restriction is to prevent your users from changing the code generation option for a VI you designed to build in a specific way. But you and your users can change the project code generation settings for a single VI in the **Source File Settings** page if code generation settings have not been set in the **VI Properties** dialog box.

For your target to support VI-specific code generation options, you must use `LEP_Utility_GetProjectVISettings.vi`, located in the `LabVIEW 8.5\vi.lib\Platform\EmbProject\utilities`

directory, to add the CCodeGenVISettings code generation attribute to the variant that determines how the LabVIEW C Code Generator generates the C code from the block diagram.

Expression Folding

Generate better performing and more efficient code by enabling expression folding, which collapses groups of nodes into single expressions that are easily recognized by C compilers. Expression folding has been added as an additional input into the VI Server call. Refer to Chapter 4, *Defining Code Generation Options*, in the *LabVIEW Embedded Development Module Porting Guide* for more information about different code generation options and the VI Server call. To enable expression folding for your target, add a checkbox on your **Build Specification Properties** dialog box.



Note You cannot debug an embedded VI while using expression folding because expression folding eliminates some wires in the generated C code.

New Embedded Development Module Functions Palette

Common utility VIs you use to implement your target are located on the **Embedded Development Module** palette. You can use these VIs only on the host computer. These VIs are not available on any target palette. Subpalettes include the following:

- Error Handling Utilities
- Debugging Utilities
- File List Utilities
- Memory Map Utilities
- Miscellaneous Utilities
- Progress Bar Utilities
- Project Utilities
- Status Window Utilities
- String & Path Utilities
- Target Server Utilities
- Telnet Utilities

Target Server

Use the Target Server to download, execute, and monitor an embedded application over a TCP connection. The Target Server provides a way to download and run an embedded application without requiring both a serial and Ethernet connection. To use the Target Server with an embedded target, the target must have a TCP connection, a TFTP client, and a telnet

server. You use a configuration cluster in the Target Server Utilities VIs to define command strings and corresponding regular expressions to parse the command output.

The Freescale M5329EVB example target uses the Target Server.

Refer to the *LabVIEW Embedded Development Module Porting Guide* for information about the Target Server.

Updated Embedded Automatic Test Framework

The Embedded automatic test framework includes many new tests.



Note Running the complete test framework can take several hours to complete.

All targets and associated tests are now in a tree in the **LabVIEW Embedded Test Framework** dialog box. In previous versions of the Embedded Development Module, dialog boxes can obscure the testing progress and results. You now see the testing progress and results directly in a panel layout in the **LabVIEW Embedded Test Framework** dialog box.

You can select whether to run all tests for an embedded target, all tests for all targets, only some of the tests, and so on. Select **Tools»Embedded Tools»Run All Embedded Tests** to open the **LabVIEW Embedded Test Framework** dialog box. The order of the targets in the `LEP_AutoTest.ini` file determines the order of the targets in the dialog box. Multiple tests for an embedded target execute in the order you select the tests. The `LEP_AutoTest.ini` file is located in the `labview\autotest` directory.

Refer to Chapter 13, *Embedded Automated Tests*, in the *LabVIEW Embedded Development Module Porting Guide*, available by selecting **Start»All Programs»National Instruments»LabVIEW 8.5»LabVIEW Manuals**, for token descriptions. The following configuration token is new in this version of the Embedded Development Module:

Token: `_Test_Description`

Description: A description of the test. The description appears in the **LabVIEW Embedded Test Framework** dialog box.

Dynamically Calling VIs on a Target

In previous versions of the Embedded Development Module, you can use the Call By Reference Node to dynamically call VIs only on remote machines. You now can use the Call By Reference Node to dynamically call VIs on the embedded target. The Static VI Reference function is now

supported. You can use the Static VI Reference function to maintain a static reference to a VI. You can configure the Static VI Reference function to output a generic or strictly typed VI reference. After you place the Static VI Reference function on a block diagram, double-click the function to display a file dialog box where you can select a VI.

Indicate which VIs in your project to call dynamically on the **Source Files** page of the **Build Specification Properties** dialog box.



Tip If you are upgrading from version 2.0, add the Static VI Reference function to your target palette. If you are referencing the common embedded menu file in your target menu file, you do not need to add the function as it has already been added for the example targets. Adding a function to a palette is slightly different than adding a VI to a palette. To add a function, copy the default core Application Control menu file (`labview\menus\Categories\Programming\appctl.mnu`) to your target menu directory. Then, reopen LabVIEW to reload the palettes and use the Menu Editor to remove unsupported VIs and functions from the **Application Control** palette. Refer to the *LabVIEW Help* for more information about the Menu Editor.

Background Debugging Thread for Instrumented Debugging

In previous versions of the Embedded Development Module, instrumented debugging is a foreground task, which can reduce performance during debugging. Instrumented debugging now runs as a background task that sends front panel updates and probe updates to the host computer only as needed and if higher-priority tasks are not running, which increases performance while debugging an embedded application. A Background Debug Thread (BDT) task receives messages from the send queue and sends those messages to the host computer. The BDT also reads messages from the host computer and updates the data for the front panel controls if the BDT receives a front panel control update message.

You can connect to and disconnect from a running embedded application during a debugging session. If you use instrumented debugging, you can connect to the target at any time. To connect to or disconnect from a target, right-click the build specification in the **Project Explorer** window and select **Connect** or **Disconnect** from the shortcut menu.

To implement the **Connect** and **Disconnect** menu items, use the **Shortcut Menu Items** tab in the Target Editor to add the menu items and create the plug-in VI. Your Connect plug-in VI must call one of the `nitargetxStartDebug` VIs located in the `labview\vi.lib\LabVIEW Targets\TargetShared` directory and on the **Debugging Utilities** palette. Your Disconnect plug-in VI must call the `nitargetDisconnectDebug` VI, which also is located in the same directory and on the palette.



Note You do not have to implement the **Connect** and **Disconnect** menu items to use debugging. You only need to implement them if you want the connect and disconnect functionality.

Newly Supported VIs and Functions

The Embedded Development Module now supports the following VIs and functions:

- Rendezvous VIs
 - Create Rendezvous
 - Destroy Rendezvous
 - Get Rendezvous Status
 - Not A Rendezvous
 - Resize Rendezvous
 - Wait at Rendezvous
- Static VI Reference function
- XML VIs and functions
 - Escape XML
 - Flatten To XML
 - Read From XML File
 - Unescape XML
 - Unflatten From XML
 - Write to XML File

Refer to the *LabVIEW Help*, available by selecting **Help»Search the LabVIEW Help**, for information about these VIs and functions.

Incremental Builds on Example Targets

The incremental build code generation option determines whether to rebuild everything every time you build an embedded VI into an embedded application. Incremental builds only overwrite C files that are older than the VIs from which they are generated. You can write and combine large applications with a `makefile` build process. If the VIs are in a `.lib`, all VIs in the `.lib` are regenerated if any of the VIs in the `.lib` change.

The Windows Console and Unix Console example targets contain the incremental build option.

Thread-Safe and Interrupt-Safe VIs

The following sections lists common VIs, functions, and variables that you most likely use to pass data between threads and if they are thread safe and/or interrupt safe.

Thread Safe

The following VIs and functions are thread safe:

- RTFIFO VIs
- Occurrences functions
- Semaphore VIs
- Rendezvous VIs

The following functions and variables are not thread safe:

- Queue Operations functions
- Notifier Operations functions
- Local variables
- Global variables

Interrupt Safe

The following VIs, functions, and variables are interrupt safe:

- RTFIFOs
- You can set occurrences, semaphores, and notifiers.
- You can use global variables to initialize a global variable and then read the variable in the interrupt. The Embedded Development Module example targets use global variables in this way, which is interrupt safe. Global variables are not interrupt-safe for communicating between interrupt and non-interrupt code.

Embedded Development Module Documentation

The Embedded Development Module includes the following documentation in addition to this document:

- The *LabVIEW Embedded Development Module Porting Guide*, available by selecting **Start»All Programs»National Instruments»LabVIEW 8.5»LabVIEW Manuals** and opening `EMB_Porting_Guide.pdf`, contains the information you need to port LabVIEW to a new target.
- The *LabVIEW Embedded Development Module Target Distribution Guide*, available by selecting **Start»All Programs»National**

Instruments»LabVIEW 8.5»LabVIEW Manuals and opening `EMB_Distribution_Guide.pdf`, contains the information you need as an OEM or VAR if you want to bundle and resell LabVIEW and your target.

- The readme file, available by selecting **Start»All Programs»National Instruments»LabVIEW 8.5»Readme** and opening `readme_EMB.html`, contains known issues.
- The *LabVIEW Help*, available by selecting **Help»Search the LabVIEW Help**, contains reference information about LabVIEW palettes, menus, tools, VIs, and functions. The *LabVIEW Help* also contains step-by-step instructions for using LabVIEW features. The *LabVIEW Help* uses **(EDM)** in the index to indicate topics specific to the Embedded Development Module. The *LabVIEW Help* uses **(Embedded Targets)** in the index to indicate topics that are relevant to all embedded targets.

Where to Go for Support

In addition to the Embedded Development Module documentation, you also have access to the LabVIEW Embedded Target Development Community, the LabVIEW Embedded Discussion Forum, and training.

- The LabVIEW Embedded Target Development Community provides a place for Embedded Development Module users to upload, share, and download plug-in VIs for various hardware platforms. You can access the community by visiting ni.com/info and typing `edmcomm`.
- The LabVIEW Embedded Discussion Forum provides a discussion forum for the Embedded Development Module and other LabVIEW Embedded products. You can access the forum by visiting ni.com/info and typing `edmforum`.
- On site training at National Instruments corporate headquarters or Web-based training available upon request.

National Instruments corporate headquarters is located at 11500 North Mopac Expressway, Austin, Texas, 78759-3504.

National Instruments also has offices located around the world to help address your support needs. For telephone support in the United States, create your service request at ni.com/support and follow the calling instructions or dial 512 795 8248. For telephone support outside the United States, contact your local branch office:

Australia 1800 300 800, Austria 43 0 662 45 79 90 0,
Belgium 32 0 2 757 00 20, Brazil 55 11 3262 3599,
Canada 800 433 3488, China 86 21 6555 7838,
Czech Republic 420 224 235 774, Denmark 45 45 76 26 00,

Finland 385 0 9 725 725 11, France 33 0 1 48 14 24 24,
Germany 49 0 89 741 31 30, India 91 80 41190000,
Israel 972 0 3 6393737, Italy 39 02 413091, Japan 81 3 5472 2970,
Korea 82 02 3451 3400, Lebanon 961 0 1 33 28 28,
Malaysia 1800 887710, Mexico 01 800 010 0793,
Netherlands 31 0 348 433 466, New Zealand 0800 553 322,
Norway 47 0 66 90 76 60, Poland 48 22 3390150,
Portugal 351 210 311 210, Russia 7 095 783 68 51,
Singapore 1800 226 5886, Slovenia 386 3 425 4200,
South Africa 27 0 11 805 8197, Spain 34 91 640 0085,
Sweden 46 0 8 587 895 00, Switzerland 41 56 200 51 51,
Taiwan 886 02 2377 2222, Thailand 662 278 6777,
United Kingdom 44 0 1635 523545

National Instruments, NI, ni.com, and LabVIEW are trademarks of National Instruments Corporation. Refer to the *Terms of Use* section on ni.com/legal for more information about National Instruments trademarks. Other product and company names mentioned herein are trademarks or trade names of their respective companies. For patents covering National Instruments products, refer to the appropriate location: **Help»Patents** in your software, the `patents.txt` file on your CD, or ni.com/patents.