

RELEASE NOTES

LabVIEW™ Embedded Development Module

Version 2.0

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.....	5
LabVIEW Embedded Development Module Example Targets.....	6
Selecting an Appropriate Example Target.....	6
Example Targets System Requirements	8
Upgrading from Version 1.0	9
Changes in the LabVIEW Development System.....	9
Newly Supported Data Types	9
Newly Supported Front Panel Controls	10
Upgrading Target Support	10
Changes to the Embedded Project API.....	11
Changes to On Chip Debugging Capabilities	12
Logging	13
Embedded Development Module Documentation	14
Where to Go for Support.....	14

Introduction

Graphical system design has emerged as a new approach to embedded system design. Graphical system design allows you to design algorithms using interactive design tools for control, signal processing, and advanced mathematics. You then can use your code with off-the-shelf hardware for functional prototypes. Graphical system design is also scalable enough to target your custom hardware for higher volume deployments. Refer to the National Instruments Web site at ni.com/design to learn more about the LabVIEW embedded design and prototyping platform.

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 basic 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* for more 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. Plug-in VIs for your target do not appear on the LabVIEW **Controls** or **Functions** palette.

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 compilation 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. You 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 you create with a utility when you switch to a different target that contains the target syntax checking for that target.

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 VIs from the palettes 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 to verify 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 desktop computer with Windows 2000/XP
- LabVIEW 8.2 Full or Professional Edition
- NI-VISA 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 8.2»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 to handle the math in the LabVIEW analysis library, which is floating-point. You can provide a gain in performance by handling the math 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 8.2, if not already installed.
3. Insert the LabVIEW Embedded Development Module CD and follow the instructions that appear on the screen.

You must activate the Embedded Development Module to obtain 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.

LabVIEW Embedded Development Module Example Targets

The Embedded Development Module includes several example targets. Use the example targets as a starting point when you create new embedded targets. Example targets are located in the following directory:



labview\Targets\NI\Embedded

Selecting an Appropriate Example Target

Select an appropriate example target to use as a template when you create a new embedded target. Use the target that is closest to the target and toolchain you are creating. If you are using a GNU C/C++-based (gcc) toolchain, you might want to use an eCos target. If you are using a VxWorks-based toolchain with different hardware, you might want to use a VxWorks subtarget. Subtargets are targets that reuse existing functionality from another target.

The Embedded Development Module includes a blank template target. This target is not intended to be an implementation example but serves as a good starting point when none of the example targets are appropriate. National Instruments recommends you use the blank target when porting LabVIEW to a new operating system.

None of the example targets are meant to be fully featured, ready to use targets. The different example targets have different implementations. 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.

The following table lists some of the implementation features for the example targets. Use this table to find an example of a feature you are implementing for your target.



Tip Refer to Chapter 2, *Example Targets*, in the *LabVIEW Embedded Development Module Porting Guide*, for more information about using example targets.

Target Name	Instrumented Debugging	On Chip Debugging	Pre-Built Runtime Library	Static Memory Model	Memory Mapping	Elemental I/O	IDE Integration
Code Generation Only	No	No	No	Yes	No	No	No
Axiom CMD565, eCos ROM Image	No	No	No	Yes	No	No	No
Axiom CMD565, eCos RAM Image	Serial	No	No	Yes	No	No	No
Unix Console	TCP	Eclipse	Yes	No	No	Simulated	Eclipse
Axiom CMD565, VxWorks RAM Image	No	iSYSTEM iC3000	No	Yes	No	No	No
Axiom CMD565, VxWorks ROM Image	No	No	No	Yes	Yes	No	No
Axiom CMD565, VxWorks Module	Serial	WindRiver WTX	No	Yes	No	No	No
VxWorks Simulation	TCP	No	No	Yes	No	No	No
Windows Console Application	TCP	No	Yes	Yes	No	Simulated	No
PHYTEC LPC229x, eCos	Serial	No	Yes	No	No	Yes	No
Spectrum Digital DSK6713, DSP/BIOS	RTDX	No	Yes	No	No	No	No

Example Targets System Requirements

The Embedded Development Module example targets have the following requirements:

Target Name	Hardware Requirements	Software Requirements
Code Generation Only	None	None
Axiom CMD565, eCos ROM Image	Axiom CMD-565 Development Board	Cygwin 1.5.x eCos 2.0 PowerPC toolchain
Axiom CMD565, eCos RAM Image	Axiom CMD-565 Development Board	Cygwin 1.5.x eCos 2.0 PowerPC toolchain
Unix Console	None	Cygwin 1.5.x with gcc package
Axiom CMD565, VxWorks RAM Image	Axiom CMD-565 Development Board iSYSTEM iC3000ActiveEmulator	Wind River Tornado 2.2.1 VxWorks 5.5.1 BSP for CMD565 iSYSTEM winIDEA
Axiom CMD565, VxWorks ROM Image	Axiom CMD-565 Development Board	Wind River Tornado 2.2.1 VxWorks 5.5.1 BSP for CMD565
Axiom CMD565, VxWorks Module	Axiom CMD-565 Development Board	Wind River Tornado 2.2.1 VxWorks 5.5.1 BSP for CMD565
VxWorks Simulation	None	Wind River Tornado 2.2.1 VxWorks 5.5.1 (Optional) ULIP Ethernet driver
Windows Console Application	None	One of the following: Visual Studio 6.0 Visual Studio .NET Visual C ++ Toolkit 2003 version 1.01 and Core SDK Platform
PHYTEC LPC229x, eCos	phyCORE-ARM7/LPC229x Rapid Development Kit GPIO Expansion Board	Cygwin 1.5.x eCos 2.0 ARM toolchain
Spectrum Digital DSK6713, DSP/BIOS	DSP Starter Kit (DSK) for the TMS320C6713	TI Code Composer Studio 3.1

Contact the respective vendors for more information about their hardware and software products.

Refer to ecos.sourceware.org/getstart.html for information about downloading and installing eCos.

Refer to gcc.gnu.org for information about downloading and installing GCC.

The VxWorks Development Kit for the LabVIEW Embedded Development Module includes the Tornado 2.2.1 integrated development environment and evaluation run-times for VxWorks 5.5.1 for the purpose of demonstrating the features, performance, and capabilities of these Wind River products in association with the Labview Embedded Development Module. Refer to windriver.com/alliances/eval-cd, click **National Instruments Evaluation CD Program**, and follow the instructions to receive the VxWorks Development Kit. Refer to windriver.com for more information about Wind River's Device Software Optimization products, including VxWorks real-time operating systems and Tornado, an integrated development environment.

Upgrading from Version 1.0

Changes in the LabVIEW Development System

The biggest change in the Embedded Development Module involves the change in the LabVIEW development environment that occurred in version 8.0 of LabVIEW. If you are unfamiliar with these changes, refer to the *LabVIEW Upgrade Notes* for version 8.0 on ni.com.

- You no longer need a separate LabVIEW Embedded Edition. The Embedded Development Module now installs on top of LabVIEW like other modules, such as the Real-Time Module, FPGA Module, or PDA Module.
- The LabVIEW directory structure has changed. Refer to the *LabVIEW Embedded Development Module Porting Guide* for more information about the LabVIEW directory structure.
- The **Project Explorer** window replaces the **Embedded Project Manager** window.
- The **Build Specification Properties** dialog box replaces the **Build Settings** dialog box. You now can use the **Target Properties** dialog box instead of adding tabs in the **Build Settings** dialog box or creating a new dialog box for target-specific settings.

Newly Supported Data Types

The Embedded Development Module now supports the following data types:

- Dynamic
- Measurement
- Timestamp
- Waveform
- Variant

Newly Supported Front Panel Controls

The Embedded Development Module now supports the following front panel controls:

- Gauge
- Timestamp
- Password display support for string controls

Upgrading Target Support

Because each target implementation is different, upgrading target support for version 8.2 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\Targets`. National Instruments recommends using some version of your company name.
2. Create an `Embedded` folder under the `labview\Targets\<company name>` directory. LabVIEW does not recognize any targets outside of this directory.
3. Copy your target directory from LabVIEW 7.1 Embedded Edition, which is located in `labview_embedded\resource\LabVIEW Targets\Embedded`, to the new LabVIEW 8.2 directory you created in steps 1 and 2.
4. Select **Tools»Embedded Tools»Target Editor** in LabVIEW to launch the Target Editor and create and modify the `TgtSupp.xml` file. The `TgtSupp.xml` file is how LabVIEW incorporates your target into LabVIEW. The `TgtSupp.xml` file replaces all VIs the `TgtSupp.ini` file references in version 1.0. Refer to the *LabVIEW Embedded Development Module Porting Guide*, available by selecting **Start»All Programs»National Instruments»LabVIEW 8.2»LabVIEW Manuals** and opening `EMB_Porting_Guide.pdf`, for more information about using the Target Editor to manage embedded targets.
5. Open all of your existing plug-in VIs in version 8.2 of LabVIEW and verify none of the VIs are broken.
6. Migrate the **Build Setting** dialog box user interface and functionality to the **Build Specification Properties** dialog box.
7. Move any target-specific VIs and palettes from LabVIEW 7.1 Embedded Edition directory to LabVIEW 8.2 directory.

Changes to the Embedded Project API

Modified VI

In the Embedded Development Module 1.0 you use the `LEP_Uilities_GetSetProjectData.vi`, shown in Figure 1.

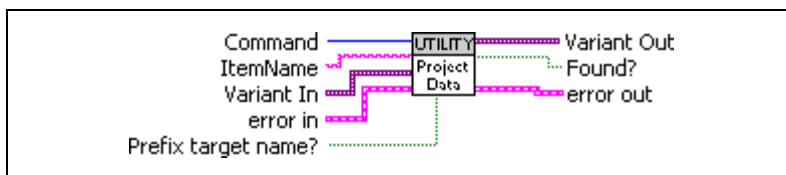


Figure 1. `LEP_Uilities_GetSetProjectData.vi`

In the Embedded Development Module 2.0 you use the `LEP_Uilities_GetSetProjectData_Variant.vi`, shown in Figure 2.

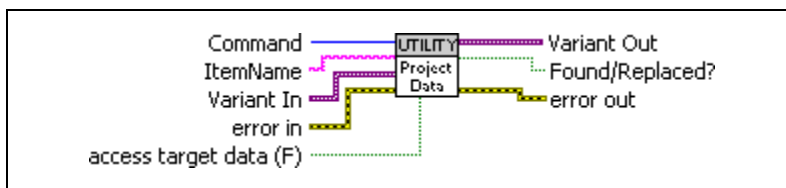


Figure 2. `LEP_Uilities_GetSetProjectData_Variant.vi`

In `LEP_Uilities_GetSetProjectData_Variant.vi` the **access target data (F)** input replaces the **Prefix target name?** input. You set the **access target data (F)** input to **TRUE** to access data for the active target project item and to **FALSE** to access data for the active build specification project item. The **access target data (F)** input is ignored if this VI is called from the target plug-in.

Deprecated VIs

The following VIs are no longer supported.

- `nitargetGetTargetFilePath.vi`—You must use `LEP_Uilities_GetTarget_Info.vi` instead.
- `SetSMASState.vi`—You must create a separate target for static memory allocation. Refer to the *LabVIEW Embedded Development Module Porting Guide* for information about static memory allocation.
- `LEP_Uilities_SwitchTarget.vi`—You can no longer use this VI because the **Execution Target** pull-down menu is no longer supported.

New VIs

The following VIs are new to the Embedded Project API.

- `LEP_Uilities_GetContext.vi`—This VI returns the context, target ID, and build ID of the active target. You can use the **Context** output to open a VI under the active target. You can call this VI from target and build specification plug-in VIs.
- `LEP_Uilities_GetTargetInfo.vi`—The **display name** output is the name of the active target in the **Project Explorer** window. The **preallocate UI Enabled** output is TRUE if static memory allocation is enabled for the active target. You can call this VI from target and build specification plug-in VIs.
- `LEP_Uilities_GetBuildSpecInfo.vi`—The **display name** output is the name of the active target in the **Project Explorer** window. This VI returns an empty string if you call it from a target plug-in VI.

Changes to On Chip Debugging Capabilities

Version 2.0 adds new features to improve the synchronization between an integrated development environment (IDE) and LabVIEW during an on chip debugging session.

New features include the following:

- **Synchronized stepping**—Stepping through code in an IDE updates the current node in LabVIEW.
- **Synchronized breakpoints**—Breakpoints set and cleared in an IDE updates the breakpoints on the corresponding nodes in LabVIEW.
- **Application crash point detection**—When an IDE sends a crash message, the faulty node highlights in LabVIEW and a message displays the reason for the crash.
- **Show source code**—A **Show Source Code** shortcut menu option is available by right-clicking a node during an on chip debugging session. The corresponding source code is displayed in the IDE.

New Plug-In VIs

The on chip debugging interface (OCDI) includes the following new plug-in VIs:

Plug-In VI	Description
IInitIndication	Called at initialization time to initialize the IDE integration occurrence. The plug-in can spawn a background VI to listen for events from the IDE.
IGetIndication	Retrieves the last event from the IDE.
IShowSource	Called when the user right-clicks a node and selects Show source code from the shortcut menu.
IShowCrash	Called when a crash event is received.

Refer to Chapter 16, *On Chip Debugging*, and Chapter 17, *Integrating LabVIEW with Another IDE*, of the *LabVIEW Embedded Development Module Porting Guide*, for more information about the new plug-in VIs.

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 your porting because you can quickly locate any 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`.

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 8.2»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 8.2»LabVIEW Manuals** and opening `EMB_Distribution_Guide.pdf`, contains the information you need as an OEM if you want to bundle and resell LabVIEW and your target.
- The readme file, available by selecting **Start»All Programs»National Instruments»LabVIEW 8.2»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 Embedded Development Module topics.

Where to Go for Support

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.