Contents

CHAPTER 11
Deploying and Replicating Systems

Application Deployment

All LabVIEW development for real-time targets and touch panel targets is done on a Windows PC. To run the code embedded on the targets, you need to deploy the applications. Real-time controllers and touch panels, much like a PC, have both volatile memory (RAM) and nonvolatile memory (hard drive). When you deploy your code, you have the option to deploy to either the volatile memory or nonvolatile memory on a target.

Deploy to Volatile Memory

If you deploy the application to the volatile memory on a target, the application does not remain on the target after you cycle power. This is useful when iteratively developing your application and testing your code.

Deploy to Nonvolatile Memory

If you deploy the application to the nonvolatile memory on a target, the application remains after you cycle the power on the target. You can set applications stored on nonvolatile memory to start up automatically when the real-time target boots. This is useful when you have finished development and want to create a stand-alone embedded system.

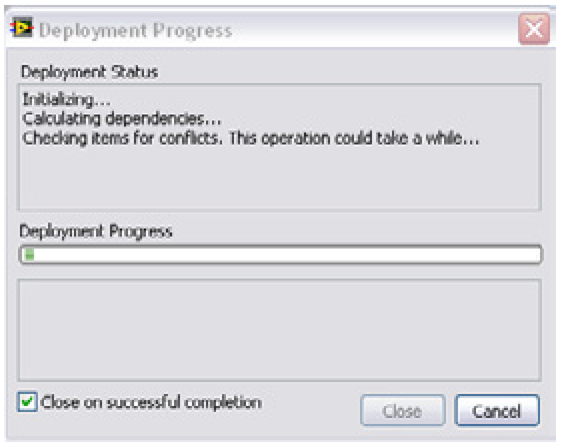
Deploying Applications to CompactRIO

Deploy a LabVIEW VI to Volatile Memory

When you deploy an application to the volatile memory of a CompactRIO controller, LabVIEW collects all of the necessary files and downloads them over Ethernet to the CompactRIO controller. To deploy an application to volatile memory you need to

- Target the CompactRIO controller in LabVIEW
- Open a VI under the controller
- Click the Run button

LabVIEW verifies that the VI and all subVIs are saved, deploys the code to the nonvolatile memory on the CompactRIO controller, and starts embedded execution of the code. Use the ability to deploy to volatile memory to quickly iterate through application development.



Deployment Progress

Deployment Status

Initializing...
Calculating dependencies...
Checking items for conflicts. This operation could take a while...

Deployment Progress

Figure 11.1. LabVIEW Deploying an Application to the Nonvolatile Memory of the Controller

Deploy a LabVIEW VI to Nonvolatile Memory

Once you have finished developing and debugging your application, you likely want to deploy your code to the nonvolatile memory on the controller so that it persists through power cycles. To deploy an application to the nonvolatile memory, you first need to build the VI into a real-time executable (RTEXE). RTEXEs can be configured to run when the real-time device boots, allowing for truly headless embedded operation.

Building an Executable From a VI

With the LabVIEW project, you can build an executable real-time application from a VI by creating a build specification under the real-time target in the LabVIEW Project Explorer. When you right-click on Build Specifications, you are presented with the option of creating a Real-Time Application along with a Source Distribution, Zip File, and so on.

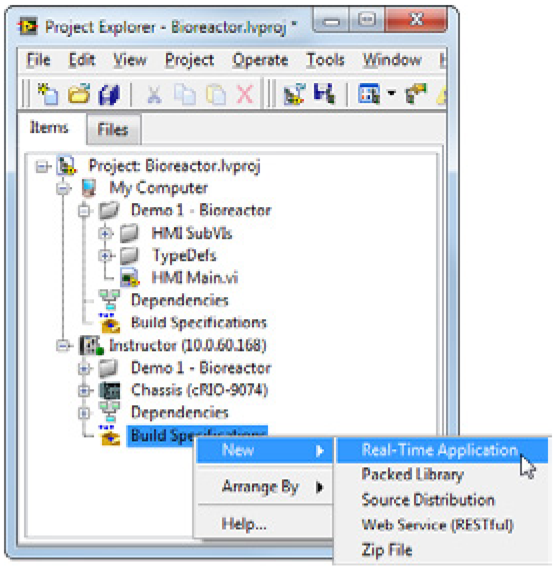
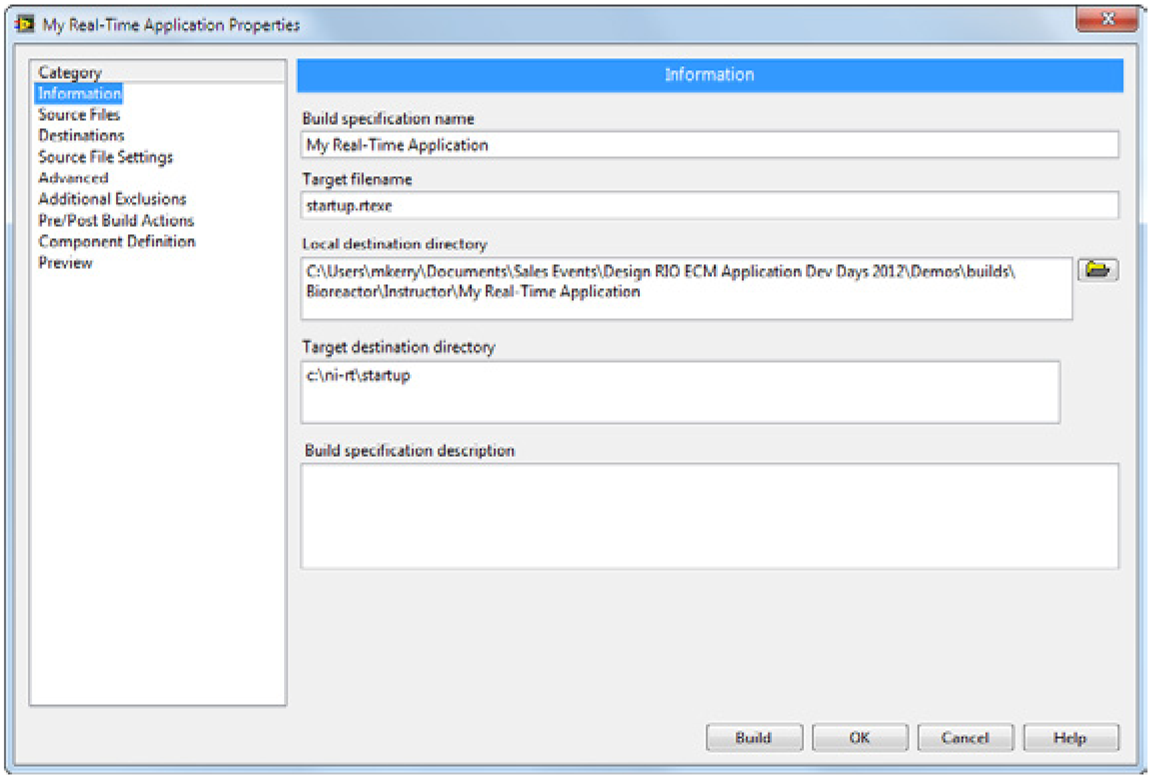


Figure 11.2. Create a new real-time application build specification.

After selecting Real-Time Application, you see a dialog box featuring two main categories that are most commonly used when building a real-time application: Information and Source Files.

The Information category contains the build specification name, executable filename, and destination directory for both the real-time target and host PC. You can change the build specification name and local destination directory to match your nomenclature and file organization. You normally do not need to change the target filename or target destination directory.


My Real-Time Application Properties

Build specification name: My Real-Time Application

Target filename: startup.rtexe

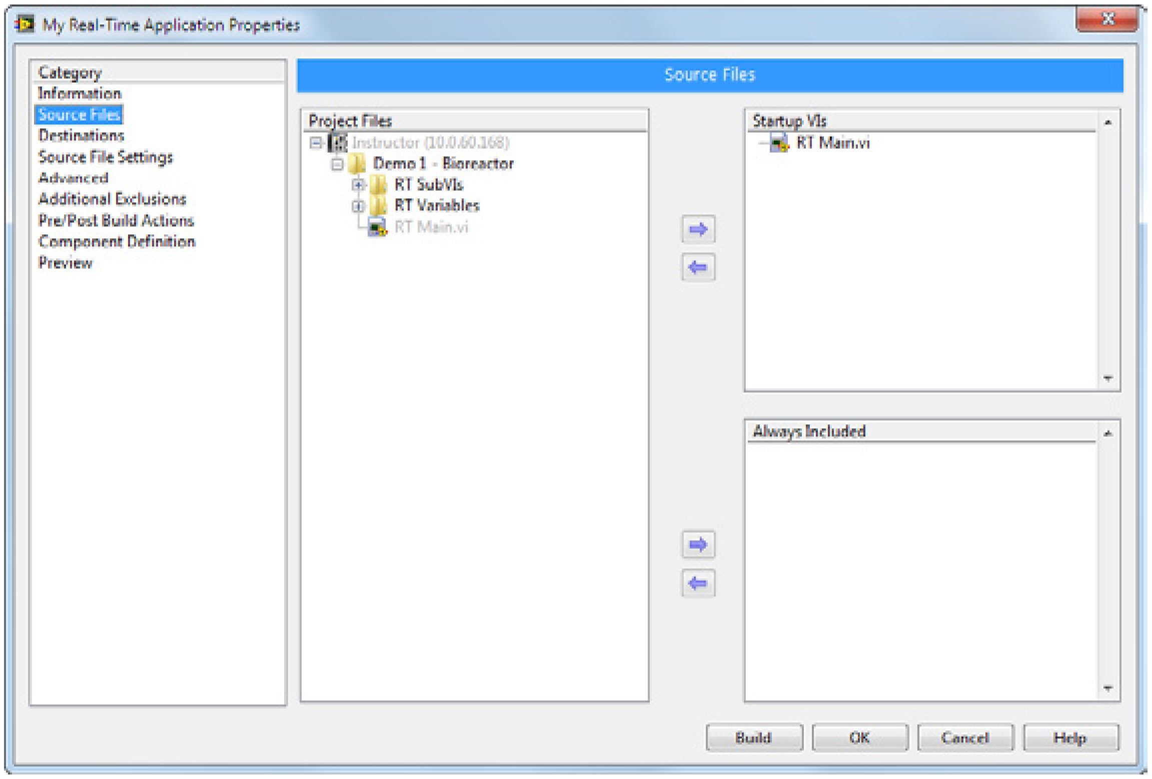
Local destination directory: C:\Users\mkerry\Documents\Sales Events\Design RIO ECM Application Dev Days 2012\Demos\builds\Bioreactor\Instructor\My Real-Time Application

Target destination directory: c:\ni-rt\startup

Build specification description:

Figure 11.3. The Information Category in the Real-Time Application Properties

The Source Files category is used to set the startup VIs and include additional VIs or support files. You need to select the top-level VI from your Project Files and set it as a Startup VI. For most applications, a single VI is chosen to be a Startup VI. You do not need to include lvlib or set subVIs as Startup VIs or Always Included unless they are called dynamically in your application.


My Real-Time Application Properties

Category
Information
Source Files
Destinations
Source File Settings
Advanced
Additional Exclusions
Pre/Post Build Actions
Component Definition
Preview

Source Files

Project Files
Instructor (10.0.60.168)
Demo 1 - Bioreactor
RT SubVIs
RT Variables
RT Main.vi

Startup VIs
RT Main.vi

Always Included

Figure 11.4. Source Files Category in the Real-Time Application Properties (In this example, the cRIOEmbeddedDataLogger (Host).vi was selected to be a Startup VI.)

After all of the options have been entered on the required category tabs, you can click OK to save the build specification or you can directly build the application by clicking the Build button. You can also right-click on a saved build specification and select Build to build the application.

When you build the application, an executable is created and saved on the hard drive of your development machine in the local destination directory.

Setting an Executable Real-Time Application to Run On Startup

After an application has been built, you can set the executable to automatically start up as soon as the controller boots. To set an executable application to start up, you should right-click the Real-Time Application option (under Build Specifications) and select Set as startup. When you deploy the executable to the real-time controller, the controller is also configured to run the application automatically when you power on or reboot the real-time target. You can select Unset as Startup to disable automatic startup.

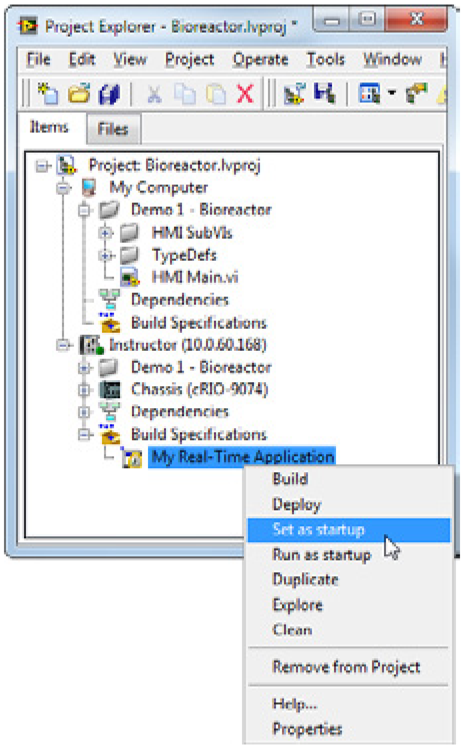


Figure 11.5. Configuring a Build Specification to Run When an Application Boots

Deploy Executable Real-Time Applications to the Nonvolatile Memory on a CompactRIO System

After configuring and building your executable, you now need to copy the executable and supporting files to the nonvolatile memory on the CompactRIO controller and configure the controller so the executable runs on startup. To copy the files and configure the controller, right-click on the Real-Time Application option and select Deploy. LabVIEW then copies the executable onto the controller's nonvolatile memory and modifies the **ni-rt.ini** file to set the executable to run on startup. If you rebuild an application or change application properties (such as configuring it not to run on startup), you must redeploy the real-time application for the changes to take effect on the real-time target.

If you used the default settings, the real-time executable (RTEXE) is located in the NI-RT\ni-rt\startup folder on the target with the name supplied in the target filename box from the Information category and the extension **.rtexe**.

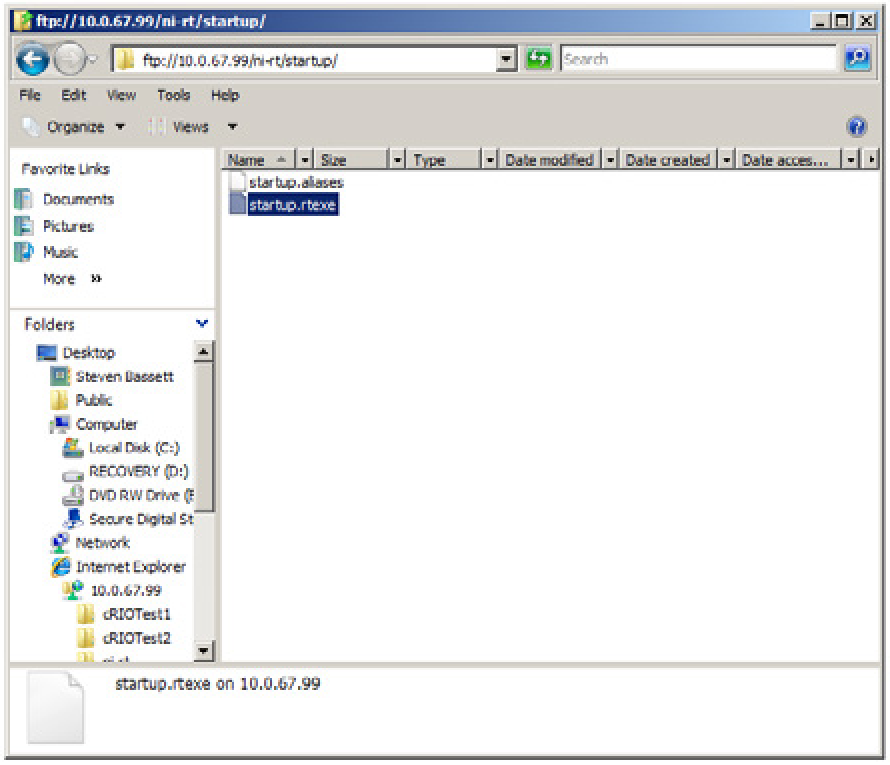


Figure 11.6. The Default Location of the startup.rtexe on a CompactRIO Controller

Deploying LabVIEW FPGA Applications

Once the development phase of the FPGA application is complete, you need to deploy the generated bitfile, also referred to as a personality, to the system. You have two main ways to embed the FPGA application onto a target with a programmable onboard FPGA. In the vast majority of applications, an Open FPGA VI Reference is used in the host application (real-time or Windows) to communicate with the FPGA application. The Open FPGA VI Reference stores the FPGA application, and is capable of deploying to the FPGA at run time. In addition to the Open FPGA VI Reference, the FPGA bitfile (representing the FPGA application) can be downloaded to nonvolatile Flash memory alongside the FPGA, and can be configured to deploy and run on the FPGA when the target is powered. The two FPGA deployment methodologies are illustrated in Figure 11.7.

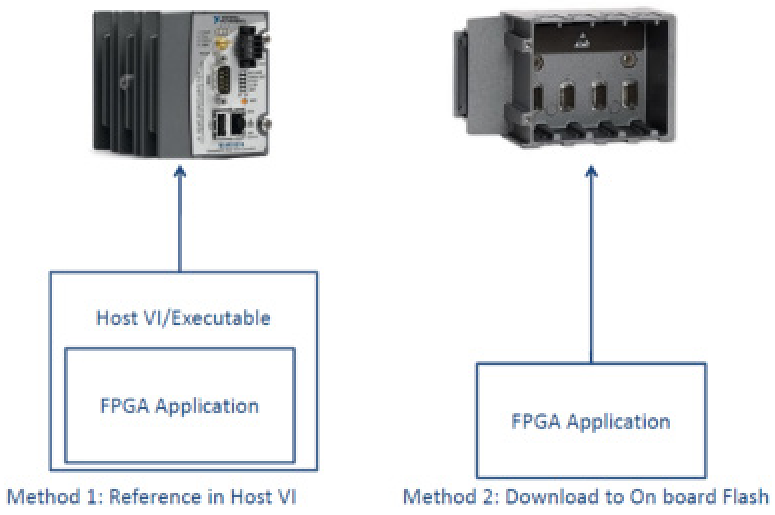

Host VI/Executable
FPGA Application
FPGA Application
Method 1: Reference in Host VI
Method 2: Download to On board Flash

Figure 11.7. Two Stand-Alone Deployment Options for LabVIEW FPGA

Method 1: Reference in Host VI

One method for deploying an FPGA personality is to embed it in the host application as shown in Figure 11.8. This can be achieved by using the Open FPGA VI Reference function in the host application. When the host application is then compiled into an executable, the FPGA application is embedded in the Open FPGA VI Reference function call. When the host application is embedded on the target and runs, the Open FPGA VI Reference function is called and then it downloads and runs the FPGA bitfile and outputs a reference to the bitfile for subsequent use in the host application. If the referenced FPGA bitfile is already running on the FPGA when the Open FPGA VI Reference is called, the function does not download the FPGA bitfile and instead only generates a reference to the bitfile for downstream use in the host application.

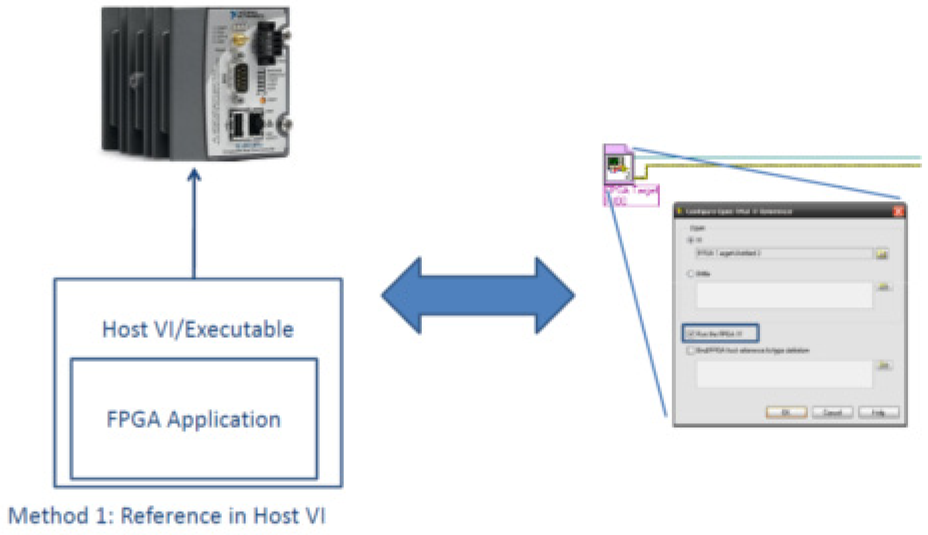


Figure 11.8. If you use an Open FPGA VI Reference in a host VI, then you embed the FPGA's bitfile in the host executable.

One caveat to relying solely on the Open FPGA VI Reference function to download and run the FPGA application is that the host application must first initialize before the FPGA application can be loaded. This causes a delay from device power up to the configuration and operation of the FPGA. On device power, the input and output lines on the FPGA are in an indeterminate state until the host application loads and executes the Open FPGA VI Reference function. The delay in loading and executing the host application can easily be on the order of a minute, posing a potential safety and reliability risk as the FPGA I/O states remain uncontrolled. Therefore, it's recommended that the FPGA bitfile implements safe states and be stored on the onboard FPGA flash memory.

Method 2: Storing the Application in Nonvolatile Flash Memory on the FPGA Target

In addition to relying solely on the Open FPGA VI Reference, you can download the FPGA bitfile to flash memory on the target device using the RIO Device Setup that is included with the NI-RIO driver as shown in Figure 11.9. You can then configure the bitfile downloaded to flash memory to load and run the personality onto the FPGA as well as load independently of the Open FPGA VI Reference function. To learn more about this download process, reference the KnowledgeBase article How Do I Download a Bitfile to My Target Without LabVIEW FPGA? You should download an FPGA bitfile implementing safe states to the flash memory and set it to run on device power up, or reboot, so that all of the input and output lines are immediately driven to a known state.

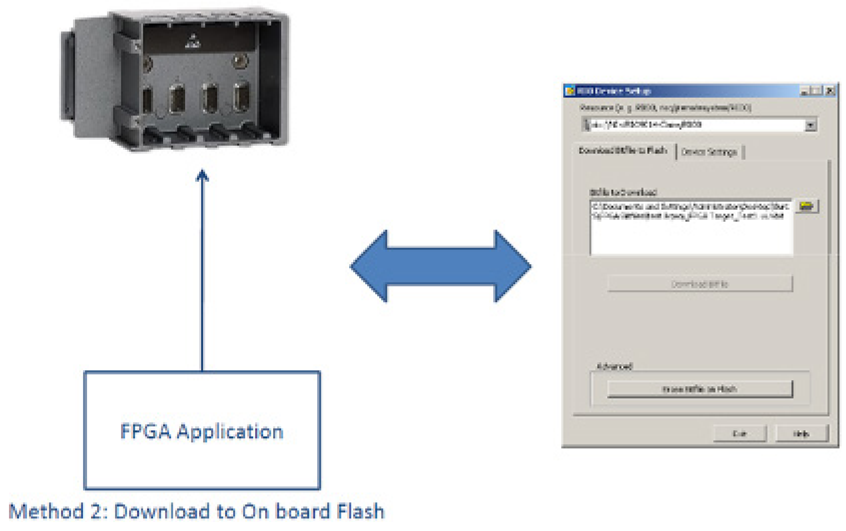
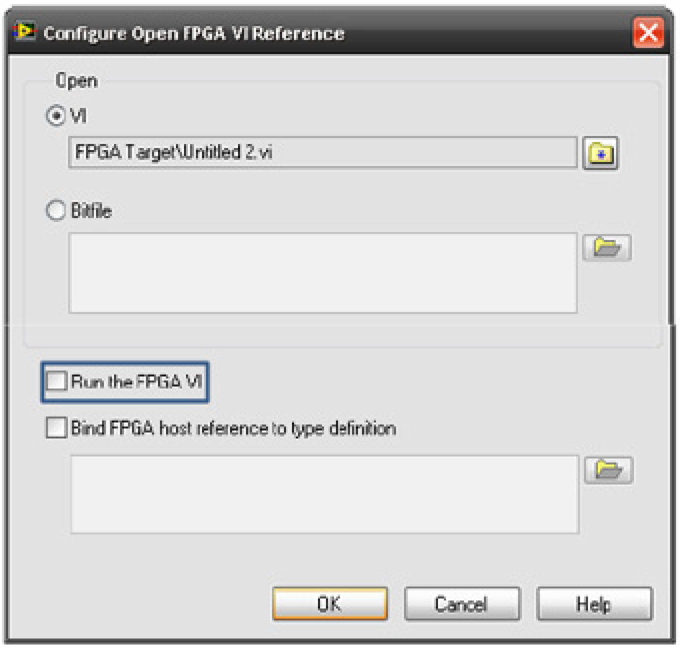


Figure 11.9. You can download the FPGA bitfile to the FPGA's onboard flash memory using the RIO Device Setup.

If you have downloaded an FPGA application to flash memory and set it to run on device power or device reboot, you can still use the Open FPGA VI Reference to communicate between the FPGA and host applications. The Open FPGA VI Reference simply generates a reference if the FPGA bitfile running on the device matches the FPGA bitfile stored within the Open FPGA VI Reference node. If, however, the bitfile stored in the Open FPGA VI Reference function differs from the bitfile running on the FPGA, note that the Run the FPGA VI replaces the bitfile running on the FPGA with the bitfile contained in the Open FPGA VI Reference function. For more information, refer to the Open FPGA VI Reference function documentation in the LabVIEW Help.


Configure Open FPGA VI Reference

Open

[✓ VI]

FPGA Target\Untitled 2.vi

Bitfile

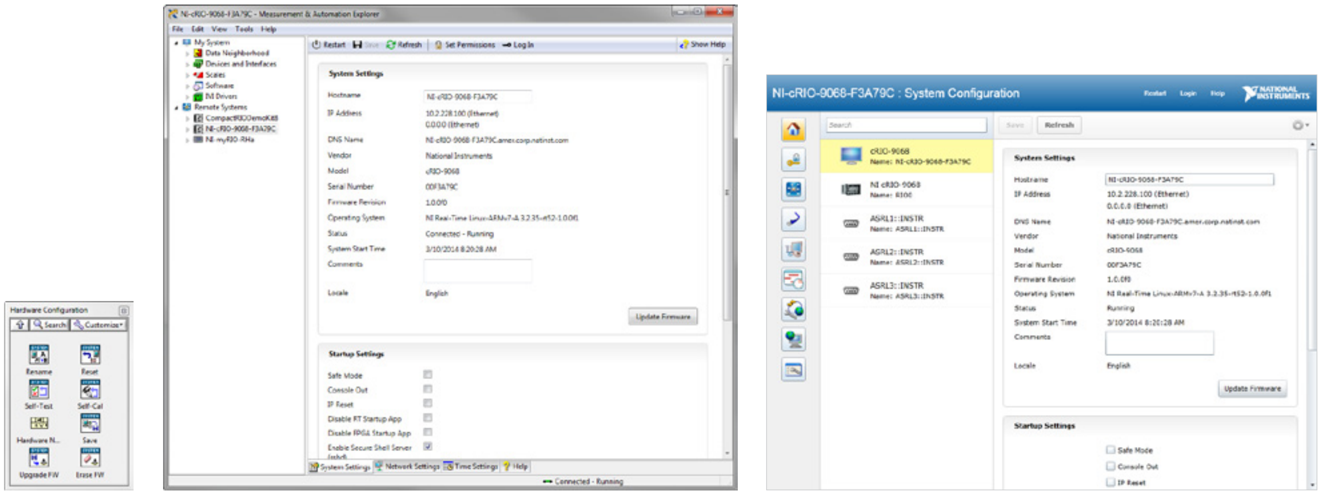
Run the FPGA VI

Bind FPGA host reference to type definition

Figure 11.10. The Run the FPGA VI option in the Open FPGA VI Reference configuration window toggles whether the Open FPGA VI Reference function overwrites the bitfile loaded to the FPGA in cases where the bitfile running on the FPGA doesn't match the bitfile embedded in the Open FPGA VI Reference function.

It is generally recommended that you embed the same FPGA personality referenced in the host application into flash memory. This is to enable greater safety and reliability by having the I/O state immediately controlled on device power or reboot. For more information on managing FPGA deployments, see the NI Developer Zone document Managing FPGA Deployments.

Beginning in LabVIEW 2013, the System Configuration API, Measurement & Automation Explorer (MAX), and the Web-Based Configuration and Monitoring interface all provide support for managing FPGA bitfiles for targets that run NI Linux Real-Time, such as the NI cRIO-9068.



System Settings	
Hostname	NI-cRIO-9068-F3A79C
IP Address	10.2.228.100 (Ethernet) 0.0.0.0 (Ethernet)
DNS Name	NI-cRIO-9068-F3A79C.amer.corp.natinst.com
Vendor	National Instruments
Model	cRIO-9068
Serial Number	00F3A79C
Firmware Revision	1.0.0f0
Operating System	NI Real-Time Linux ARMv7-A 3.2.35-rt52-1.0.0f1
Status	Connected - Running
System Start Time	3/10/2014 8:20:28 AM
Comments	
Locale	English
Startup Settings	
Safe Mode	☐
Console Out	☐
IP Reset	☐
Disable RT Startup App	☐
Disable FPGA Startup App	☐
Enable Secure Shell Server (sshd)	✓

Connected - Running

cRIO-9068
Name: NI-cRIO-9068-F3A79C

NI cRIO-9068
Name: RIO0

ASRL1::INSTR
Name: ASRL1::INSTR

ASRL2::INSTR
Name: ASRL2::INSTR

ASRL3::INSTR
Name: ASRL3::INSTR

System Settings	
Hostname	NI-cRIO-9068-F3A79C
IP Address	10.2.228.100 (Ethernet) 0.0.0.0 (Ethernet)
DNS Name	NI-cRIO-9068-F3A79C.amer.corp.natinst.com
Vendor	National Instruments
Model	cRIO-9068
Serial Number	00F3A79C
Firmware Revision	1.0.0f0
Operating System	NI Real-Time Linux ARMv7-A 3.2.35-rt52-1.0.0f1
Status	Running
System Start Time	3/10/2014 8:20:28 AM
Comments	
Locale	English
Startup Settings	
☐ Safe Mode	
☐ Console Out	
☐ IP Reset	

Figure 11.11. In LabVIEW 2013, FPGA flash deployment on NI Linux Real-Time targets is supported through the System Configuration API, MAX, and the Web-Based Configuration and Monitoring interface.

Deploying Applications That Use Network-Published Shared Variables

The term *network shared variable* refers to a software item on the network that can communicate between programs, applications, remote computers, and hardware. Find more information on network shared variables in Chapter 4: Best Practices for Network Communication.

You can choose from two methods to explicitly deploy a shared variable library to a target device:

1. You can target the CompactRIO system in the LabVIEW project, place the library below the device, and deploy the library. This writes information to the nonvolatile memory on the CompactRIO controller and causes the Shared Variable Engine to create new data items on the network.

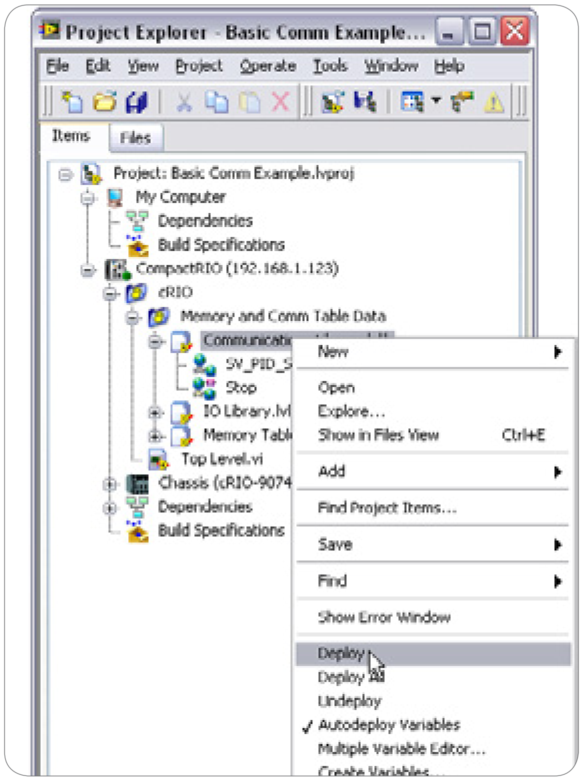


Figure 11.12. Deploy libraries to real-time targets by selecting Deploy from the right-click menu.

2. You can programmatically deploy the library from a LabVIEW application running on Windows using the Application Invoke Node.

   - On the block diagram, right-click to open the programming palette, go to **Programming»Application Control,** and place the Application **Invoke Node** on the block diagram.

   - Using the hand tool, click on **Method** and select **Library»Deploy Library.**

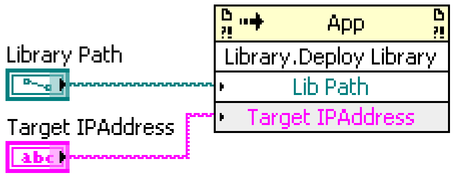


Figure 11.13. You can programmatically deploy libraries to real-time targets using the Application Invoke Node on a PC.

   - Use the Path input of the Deploy Library Invoke Node to point to the library(s) containing your shared variables. Also specify the IP address of the real-time target using the Target IP Address input.

Undeploy a Network Shared Variable Library

Once you deploy a library to a Shared Variable Engine, those settings persist until you manually undeploy them. To undeploy a library

3. Launch the NI Distributed System Manager (from **LabVIEW»Tools** or from the Start Menu).

4. Add the real-time system to My Systems (**Actions»Add System to My Systems**).

5. Right-click on the library you wish to undeploy and select **Remove Process**.

Deploy Applications That Are Shared Variable Clients

Running an executable that is only a shared variable client (not a host) does not require any special deployment steps to deploy libraries. However, the controller does need a way to translate the name of the system that is hosting the variable into the IP address of the system that is hosting the variable.

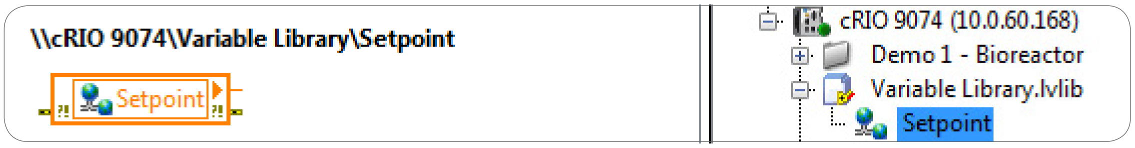
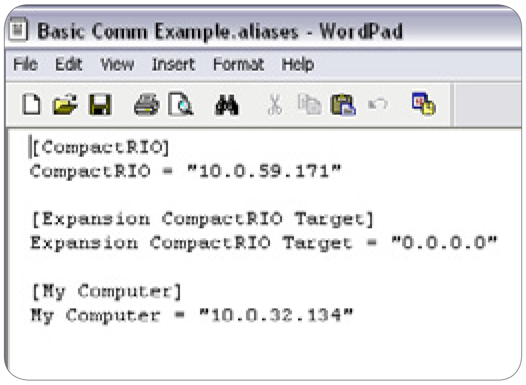


Figure 11.14. Network Variable Node and Its Network Path

To provide scalability, this information is not hard-coded into the executable. Instead, this information is stored in a file on the target called an alias file. An alias file is a human-readable file that lists the logical name of a target (CompactRIO) and the IP address for the target (10.0.62.67). When the executable runs, it reads the alias file and replaces the logical name with the IP address. If you later change the IP addresses of deployed systems, you need to edit only the alias file to relink the two devices. For real-time and Windows XP Embedded targets, the build specification for each system deployment automatically downloads the alias file. For Windows CE targets, you need to configure the build specification to download the alias file.


Basic Comm Example.aliases - WordPad
File Edit View Insert Format Help

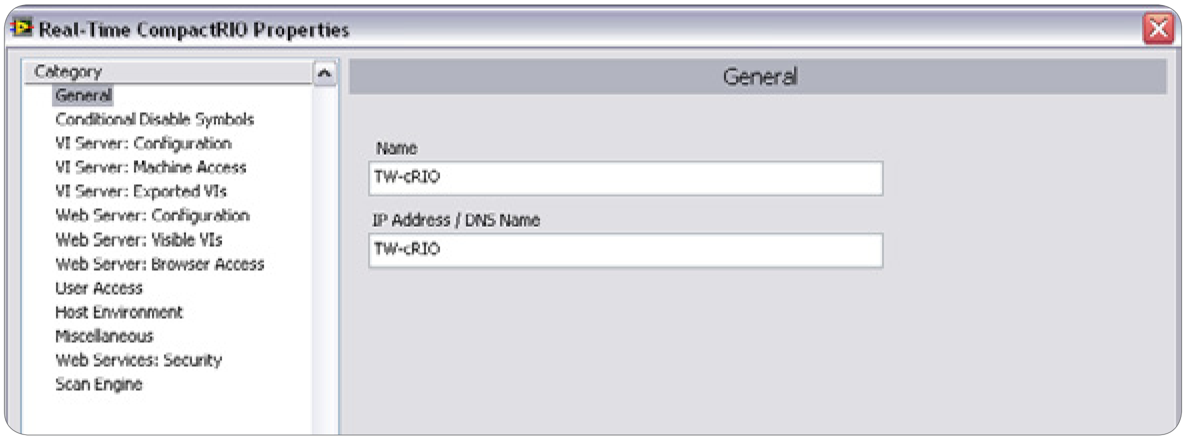
```
[CompactRIO]
CompactRIO = "10.0.59.171"

[Expansion CompactRIO Target]
Expansion CompactRIO Target = "0.0.0.0"

[My Computer]
My Computer = "10.0.32.134"
```

Figure 11.15. The alias file is a human-readable file that lists the target name and IP address.

If you are deploying systems with dynamic IP addresses using DHCP, you can use the domain name system (DNS) name instead of the IP address. In the LabVIEW project, you can type the DNS name instead of the IP address in the properties page of the target.



Name: TW-cRIO

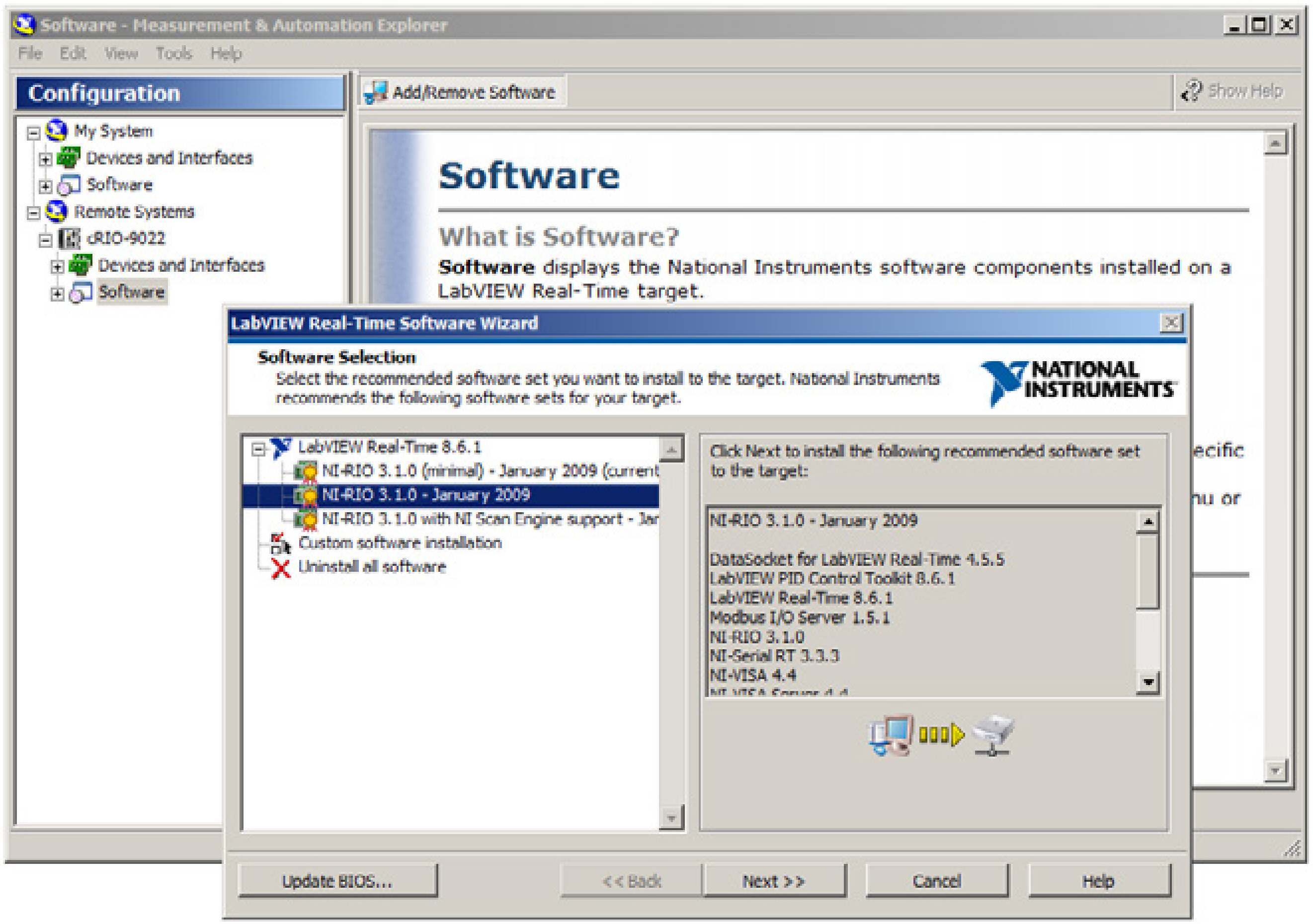
IP Address / DNS Name: TW-cRIO

Figure 11.16. For systems using DHCP, you can enter the DNS name instead of the IP address.

One good approach if you need scalability is to develop using a generic target machine (you can develop for remote machines that do not exist) with a name indicating its purpose in the application. Then as part of the installer, you can run an executable that either prompts the user for the IP addresses for the remote machine and My Computer or pulls them from another source such as a database. Then the executable can modify the aliases file to reflect these changes.

Recommended Software Stacks for CompactRIO

NI also provides several sets of commonly used drivers called recommended software sets. You can install recommended software sets on CompactRIO controllers from MAX.



Software

What is Software?

Software displays the National Instruments software components installed on a LabVIEW Real-Time target.

Software Selection

Select the recommended software set you want to install to the target. National Instruments recommends the following software sets for your target.

LabVIEW Real-Time 8.6.1
- NI-RIO 3.1.0 (minimal) - January 2009 (current
- NI-RIO 3.1.0 - January 2009
- NI-RIO 3.1.0 with NI Scan Engine support - Jar

Custom software installation

Uninstall all software

Click Next to install the following recommended software set to the target:

NI-RIO 3.1.0 - January 2009

DataSocket for LabVIEW Real-Time 4.5.5
LabVIEW PID Control Toolkit 8.6.1
LabVIEW Real-Time 8.6.1
Modbus I/O Server 1.5.1
NI-RIO 3.1.0
NI-Serial RT 3.3.3
NI-VISA 4.4

Figure 11.17. Recommended Software Sets Being Installed on a CompactRIO Controller

Recommended software sets guarantee that an application has the same set of underlying drivers for every real-time system that has the same software set. In general, there is a minimal and full software set.

Deployment Beyond the LabVIEW Project

Beyond the LabVIEW project, you can choose from two main techniques for deploying to embedded targets. These techniques, imaging and application components respectively, are far more scalable than LabVIEW Project based deployment. Imaging relies on copying the hard drive of one embedded system and propagating the copy to other systems, making it a useful technique when initially replicating embedded systems. Application components alternatively encapsulate just the real-time executable (RTEXE), and provide an effective means to perform small system upgrades to deployed systems. In addition to application components and images, deployed embedded systems often rely on local system critical files (example: application configurations). An efficient configuration management solution for an embedded system will provide for tracking and updating system critical files.

Imaging

An image in the context of NI embedded hardware and software is a single file which can contain, at a maximum, a copy of all of the files on an embedded target's disk. Given that the RTEXE, system drivers, network settings, shared variables, scan engine settings, web services, and any application configurations or plug-ins are all stored as files on disk for an embedded system, Imaging is an effective way to create a backup of a system and to replicate a given embedded system.

In order to leverage imaging, a fully configured embedded system is required. Often, this means relying on LabVIEW Project based deployment to properly deploy to the first embedded system, after which imaging can be used to setup additional systems. Given that Imaging relies on copying all the files on a properly configured embedded target, there are a few key caveats to be aware of.

The most significant caveat is that imaging can only be used to deploy/replicate to other embedded targets that are of the same model. For example, if a cRIO 9074 is used to get an image, when applying the image to setup subsequent systems, only cRIO 9074s can be used. The image from a cRIO 9074 cannot be applied to a different model cRIO, such an NI 9024. Beyond being restricted to the same model, imaging doesn't copy the FPGA flash settings between controllers as it only copies the disk that belongs to the real-time operating system on the embedded device. When applying an image, users need to take precautions to configure the imaging settings properly to preserve existing configuration or log files on the embedded target, as there is a risk overwriting existing files on the target.

Imaging is recommended when deploying to multiple targets, and is especially recommended when initially setting up large numbers of embedded targets for field deployment. Imaging is also recommended for creating restore points or backups for embedded applications. To begin using imaging to manage large numbers of embedded deployments, please refer to the following Systems Engineering utility: Replication and Deployment (RAD) Utility. The RAD utility makes use of the System Configuration API which can be used to create and deploy images. It's recommended to use the source code provided for the RAD as a starting point when attempting to create a custom imaging utility as opposed to starting from a blank VI.

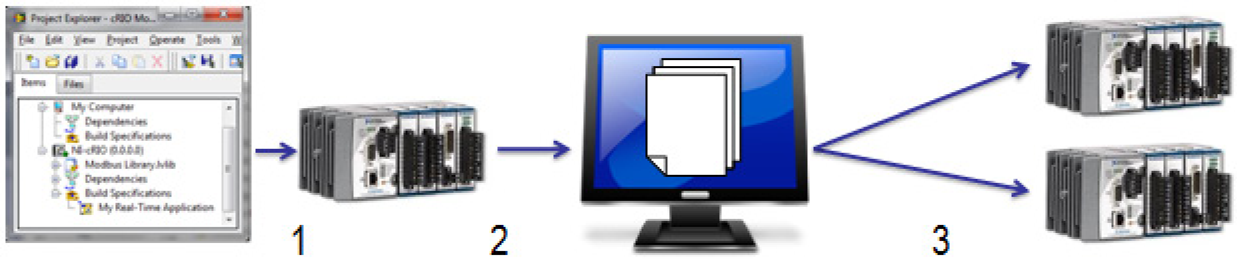


Figure 11.18. By relying on the System Configuration API and the imaging tools it provides, you can deploy images to multiple real-time targets.

Starting in LabVIEW 2013, system deployment with system images has been improved greatly on NI Linux Real-Time targets such as the cRIO-9068. These targets have the ability to run the 'Set Image' function from the System Configuration API directly, allowing for intelligent self-updating. Targets can be programmed to pull system images down from a network or locally attached storage and allow you to more securely manage larger scale deployments.

Application Components

Application components, unlike images, only encapsulate the real-time application (RTEXE) and associated driver dependencies. To better understand application components, it's useful to first cover driver components. When going through the Add/Remove Software dialog for a real-time target such as a CompactRIO, the list of options that appear both under the Recommended Software Set and the Custom Software Installation are known as components. Example components for the CompactRIO are System State Publisher, Network Variable Engine, and NI-RIO I/O Scan. Application components are similar to the driver components, but encapsulate the RTEXE, and can have driver components listed as dependencies.

Since application components only encapsulate the RTEXE and do not contain other configuration information about the embedded target, they are not a reliable way to define a backup or restore point for an embedded target. However, this same characteristic allows for them to install across any real-time target, much like driver components. Key benefits of application components are that they can be installed to other targets, and deploy much more quickly than images. For example, a real-time application developed on a cRIO 9074 can be built into an application component and deployed to an NI 9024. Application components can also be built directly from the LabVIEW Project, and unlike images, do not require an initial deployment and retrieval. To learn more about configuring and using application components, please refer to: Using Application Components to Deploy LabVIEW Real-Time Applications.

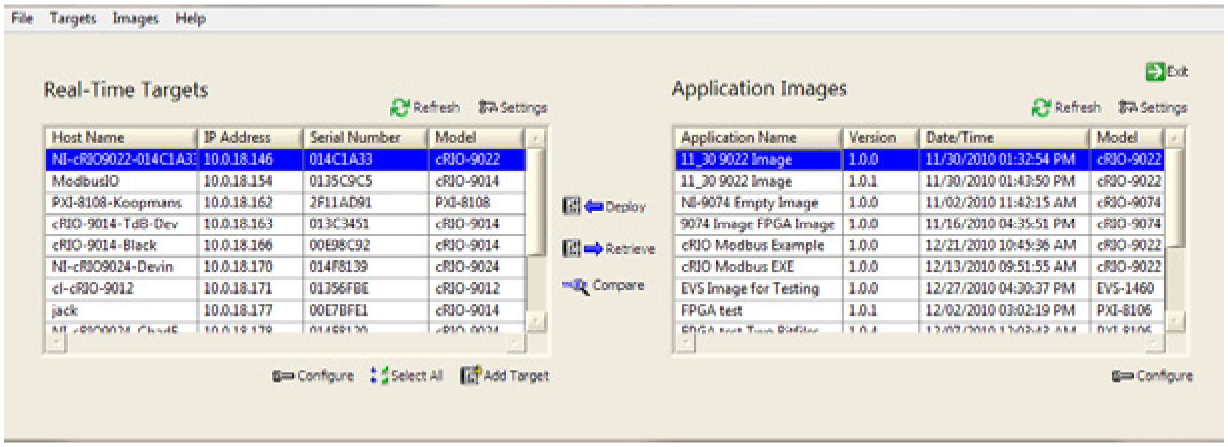
The Replication and Deployment Utility (RAD)

The RAD utility is an imaging-based, turnkey application for deploying and replicating LabVIEW Real-Time applications. It is one of the easiest and most effective ways to distribute and manage updates to embedded systems. OEMs may use this type of utility as part of their factory installation processes when assembling their products, or NI can provide a modified version of the RAD executable to end clients as a tool for performing local system updates.

You can download this utility from the NI Developer Zone document at: Replication and Deployment (RAD) Utility. After installation, you can start the RAD directly from the Tools menu in the LabVIEW environment.

When replicating a given system, an image is retrieved from the given real-time target and applied to other systems. The application image captures the contents of the hard drive of the real-time target that is fully configured with the real-time application. In addition, the RAD utility also stores information about the FPGA bitfile (if any), which is set to deploy to the FPGA flash memory. Note again that since the RAD utility relies on imaging, it can only replicate images to identical controllers and backplanes. **If you have an image from a specific controller, you can deploy that image only to controllers with the same model number.**

The main UI of the RAD utility shows two columns, listing Real-Time Targets and local Application Images.



Real-Time Targets

Host Name	IP Address	Serial Number	Model
NI-cRIO9022-014C1A33	10.0.18.146	014C1A33	cRIO-9022
ModbusIO	10.0.18.154	0135C9C5	cRIO-9014
PXI-8108-Koopmans	10.0.18.162	2F11AD91	PXI-8108
cRIO-9014-TdB-Dev	10.0.18.163	013C3451	cRIO-9014
cRIO-9014-Black	10.0.18.166	00E98C92	cRIO-9014
NI-cRIO9024-Devin	10.0.18.170	014F8139	cRIO-9024
cl-cRIO-9012	10.0.18.171	01356FBE	cRIO-9012
jack	10.0.18.177	00E7BFE1	cRIO-9014

Application Images

Application Name	Version	Date/Time	Model
11_30 9022 Image	1.0.0	11/30/2010 01:32:54 PM	cRIO-9022
11_30 9022 Image	1.0.1	11/30/2010 01:43:50 PM	cRIO-9022
NI-9074 Empty Image	1.0.0	11/02/2010 11:42:15 AM	cRIO-9074
9074 Image FPGA Image	1.0.0	11/16/2010 04:35:51 PM	cRIO-9074
cRIO Modbus Example	1.0.0	12/21/2010 10:45:36 AM	cRIO-9022
cRIO Modbus EXE	1.0.0	12/13/2010 09:51:55 AM	cRIO-9022
EVS Image for Testing	1.0.0	12/27/2010 04:30:37 PM	EVS-1460
FPGA test	1.0.1	12/02/2010 03:02:19 PM	PXI-8106

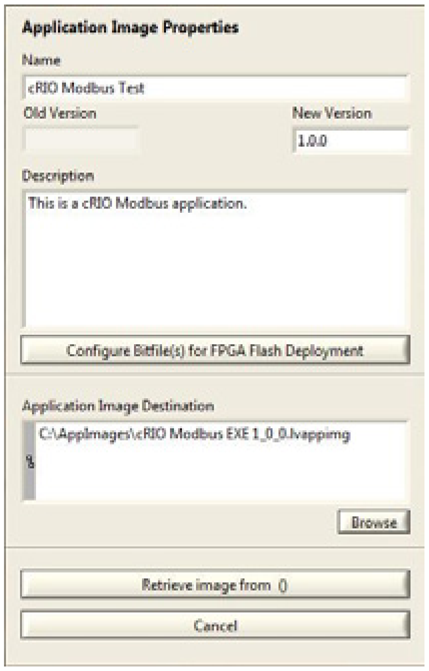
Figure 11.19. With the NI Replication and Deployment Utility, you can quickly and easily deploy images to multiple real-time targets.

The Real-Time Targets table shows all of the targets on the local subnet as well as any network targets manually added to the list. You can use these targets for both image retrieval and image deployment. The Application Images table shows all of the images that are stored on the local hard drive that can be deployed to target systems.

Retrieving Application Images

To copy the application image from a real-time target to the local hard drive, select the appropriate target in the table and click the Retrieve button in the center of the UI. You can retrieve an application image from a target in three ways. The application could either be a brand new application or a new version of a previous image either currently on the controller or previously saved to disk. If the current application is a new version of an older application, it is best to inherit the old application properties. If you are creating an image of the application for the first time, select New Application Image.

After making this selection, you are presented with the dialog box in Figure 11.20 to specify the local file for the application image and specify some additional information that is stored with the application image. If this is not a new image, properties from the old version of the image are automatically populated.



Application Image Properties

Name: cRIO Modbus Test

Old Version:

New Version: 1.0.0

Description: This is a cRIO Modbus application.

Application Image Destination: C:\AppImages\cRIO Modbus EXE 1_0_0.lvappimg

Figure 11.20. Configuring Your Application Image Properties

In addition to retrieving and deploying an image on the real-time hard drive, the utility can deploy bitfiles to FPGA flash memory. Saving a bitfile in flash memory has some advantages over deploying the bitfile from the RTEXE. For example, the host application is required to initialize before the FPGA application is loaded and, as a result, there is a delay from device power up to the configuration of the FPGA. In addition, on power up, the state of the input and output lines of your target is unknown since the FPGA has not yet been configured. Therefore, if the FPGA is completely independent of the host application, its personality should be stored in the onboard FPGA flash memory.

Using the RTAD utility, you can now save bitfiles with an image during retrieval and then later deploy them to flash memory when you deploy the image. Click Configure Bitfile(s) for FPGA Flash Deployment to edit your FPGA flash deployment settings.

Deploying Application Images

To deploy an application image to one or more targets, select the image in the right table and select the desired real-time targets in the left table. You can select multiple real-time targets using <Ctrl-Click> or <Shift-Click> or by clicking the Select All button. After completing your selection, click on Deploy at the center of the UI. A dialog confirms your real-time target selection and allows for additional target configuration options.

After verifying the selected targets and setting any desired network configuration options for your real-time targets, click on Deploy Application Image to Listed Targets to begin the deployment process. The application image is deployed to the selected targets in sequential order, one at a time. During this process, the Figure 11.21 progress dialog is shown. This process takes several minutes per real-time target.

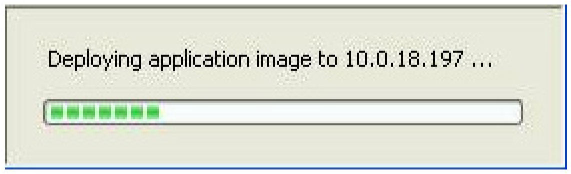
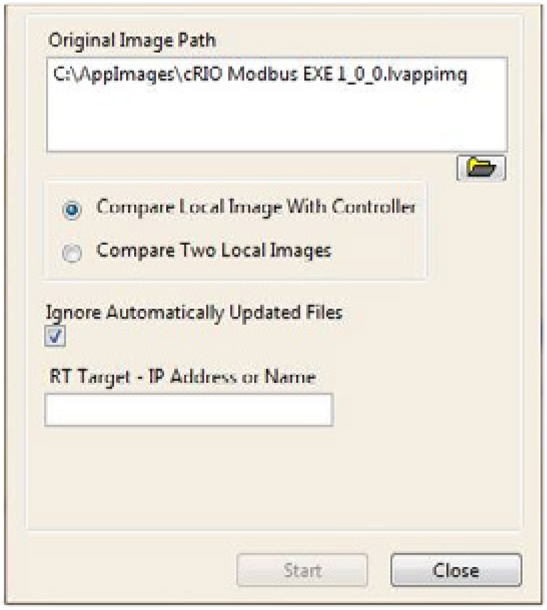

Deploying application image to 10.0.18.197 ...

Figure 11.21. Application Image Deployment Process

Comparing Application Images

In addition to retrieving and deploying application images, the utility offers an image comparison feature. The comparison tool compares each file in two images to notify you if any files have been modified, added, or removed. You can choose to compare a local image with an image on a controller, or two local images.


Original Image Path

C:\AppImages\cRIO Modbus EXE 1_0_0.lvappimg

[✓ Compare Local Image With Controller]

Compare Two Local Images

Ignore Automatically Updated Files ✓

RT Target - IP Address or Name

Figure 11.22. Compare a local image to network controller dialog.

Click the Start button to begin the comparison process. Once the comparison begins, each file on both images is compared for changes. Files that are found in one image and not the other are also identified. This process may take several minutes to complete, and a progress dialog is displayed until completion. If the tool completes with no differences found, the Figure 11.23 dialog appears.

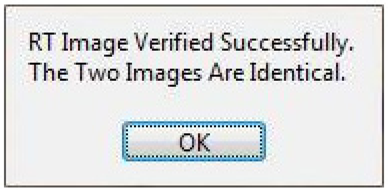

RT Image Verified Successfully.
The Two Images Are Identical.

Figure 11.23. Identical Images Notification

However, if any differences are identified, they are listed in Table 11.1. You can then log the results to a TSV file.

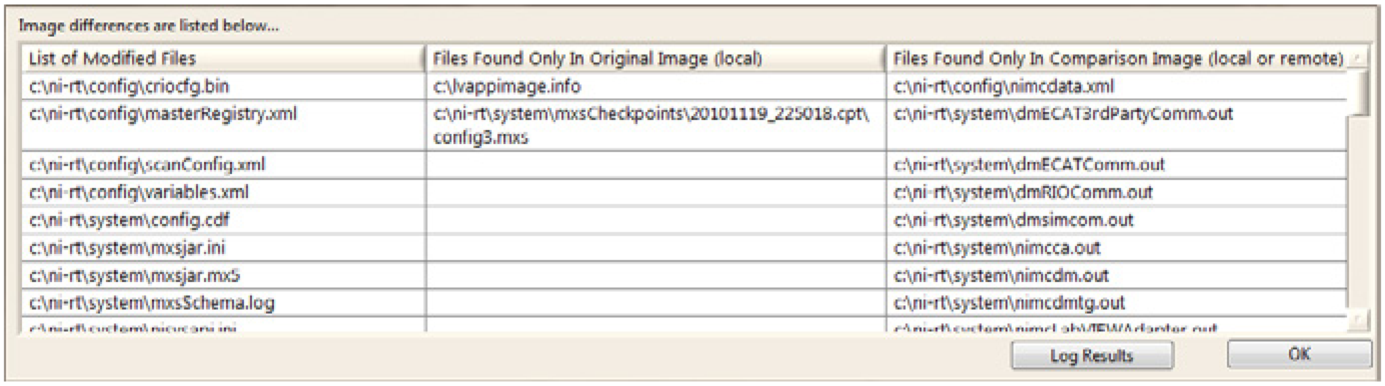


Image differences are listed below...

List of Modified Files	Files Found Only In Original Image (local)	Files Found Only In Comparison Image (local or remote)
c:\ni-rt\config\criocfg.bin	c:\lvappimage.info	c:\ni-rt\config\nimcdata.xml
c:\ni-rt\config\masterRegistry.xml	c:\ni-rt\system\mxsCheckpoints\20101119_225018.cpt\config3.mxs	c:\ni-rt\system\dmECAT3rdPartyComm.out
c:\ni-rt\config\scanConfig.xml		c:\ni-rt\system\dmECATComm.out
c:\ni-rt\config\variables.xml		c:\ni-rt\system\dmRIOComm.out
c:\ni-rt\system\config.cdf		c:\ni-rt\system\dmsimcom.out
c:\ni-rt\system\mxsjar.ini		c:\ni-rt\system\nimcca.out
c:\ni-rt\system\mxsjar.mx5		c:\ni-rt\system\nimcdm.out
c:\ni-rt\system\mxsSchema.log		c:\ni-rt\system\nimcdmtg.out
[illegible]		[illegible]

Table 11.1. Image Comparison Results

Deploying CompactRIO Application Updates Using a USB Memory Stick

If your CompactRIO systems are not available on the network, you may want to deploy an image using a USB stick. The process of updating or deploying a new application image from a USB memory device to a CompactRIO controller is based on a simple file copy operation that replaces the files on the hard drive of the CompactRIO controller with files stored on the USB memory device. This file copy operation is added as a VI to the main LabVIEW Real-Time application deployed to the controller.

A LabVIEW Real-Time application once loaded and running on the controller is stored completely in the active memory (RAM) of the controller. Because of this, you can delete and replace the application files stored on the controller hard drive while the application is running in memory. The new application becomes active by performing a reboot operation on the controller and loading the new application during the boot process.

To update the deployed code from the USB memory device in the future, you must add code to the main application that handles the deployment process. The Deploy Image.vi shown in Figure 11.24 handles the update process.

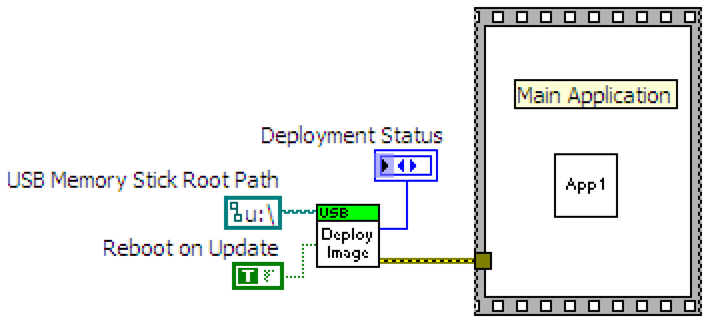


Figure 11.24. Include the USB Deploy Image VI in your deployed real-time application to enable updates from a USB memory stick.

Once you have added this code, you can build the main application into an executable and deploy it to a CompactRIO controller. The deployed application executable, together with all of the other files on the CompactRIO controller hard drive, becomes the application image. For more information on this utility including downloadable files, see the NI Developer Zone document **Reference Design for Deploying CompactRIO Application Updates Using a USB Memory Device**.

APIs for Developing Custom Imaging Utilities

You can choose from several imaging APIs—all with nearly identical functionality—depending on the version of LabVIEW you are using.

- **System Configuration API**—Recommended for LabVIEW 2011 or later
- **RT Utilities API**—Recommended only for LabVIEW 2009 and 2010
- **NI System Replication Tools**—Recommended for LabVIEW 8.6 or earlier

Another useful API discussed in this section is the CompactRIO Information Library (CRI), which you can use with the three APIs listed above to return information about the current hardware configuration of a CompactRIO system.

System Configuration API

The System Configuration API exposes MAX features programmatically. For example, you can use this API to programmatically install software sets, run self-tests, and apply network settings in addition to image retrieval and deployment. This API is located in the Functions palette under **Measurement I/O»System Configuration»Real-Time Software**.

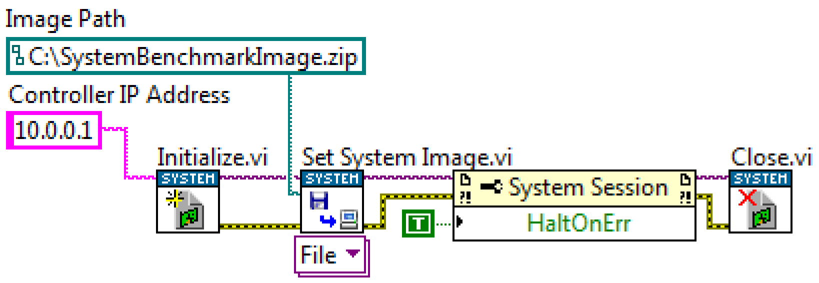


Figure 11.25. Programmatically Setting a System Image Using the System Configuration API

RT Utilities API

The RT Utilities API also includes VIs for image retrieval and deployment. It was deprecated in LabVIEW 2011 and is not recommended for new designs. The functionality it offers has been fully absorbed into the System Configuration API

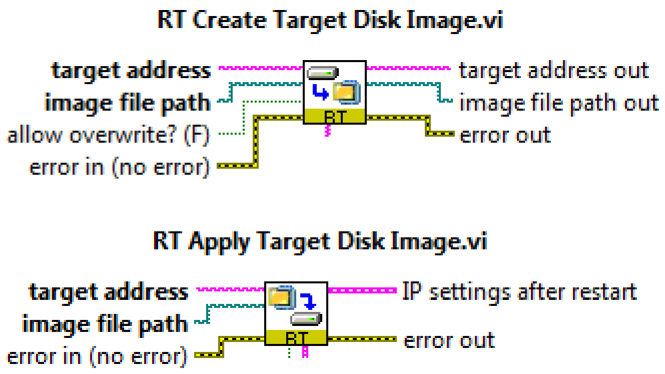


Figure 11.26. You also can use the RT Utilities VIs in LabVIEW 2009 or later to programmatically deploy and retrieve a system image.

LabVIEW Real-Time System Replication Tools

For LabVIEW 8.6 and earlier, neither the RT Utilities API nor the System Configuration API is available. For these applications, NI recommends using LabVIEW Real-Time system replication tools. For more information on these tools, including the downloadable installation files, see the NI Developer Zone document Real-Time Target System Replication.

CompactRIO Information Library (CRI)

You can use the CRI with the three APIs discussed above to detect the current configuration of a CompactRIO system. The CompactRIO Information component provides VIs to retrieve information about a local or remote CompactRIO controller, backplane, and modules including the type and serial number of each of these system components.

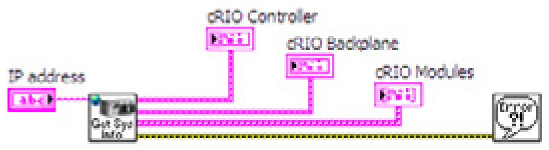


Figure 11.27. CRI Get Remote cRIO System Info.vi

You can download this library from the NI Developer Zone document
Reference Library for Reading CompactRIO System Configuration Information.

IP Protection

Intellectual property (IP) in this context refers to any unique software or application algorithm(s) that you have or your company has independently developed. This can be a specific control algorithm or a full-scale deployed application. IP normally takes a lot of time to develop and gives companies a way to differentiate from the competition; therefore, protecting this software IP is important. LabVIEW development tools and CompactRIO provide the ability to protect and lock your IP. In general, you can implement two levels of IP protection:

Lock Algorithms or Code to Prevent IP From Being Copied or Modified

If you have created algorithms for a specific function, such as performing advanced control functions, implementing custom filtering, and so on, you may want to distribute the algorithm as a subVI but prevent someone from viewing or modifying that actual algorithm. This may be to achieve IP protection or to reduce a support burden by preventing other parties from modifying and breaking your algorithms.

Lock Code to Specific Hardware to Prevent IP From Being Replicated

Use this method if you want to ensure that a competitor cannot replicate your system by running your code on another CompactRIO system or if you want your customers to come back to you for service and support.

Locking Algorithms or Code to Prevent Copying or Modification

Protect Deployed Code

LabVIEW is designed to protect all deployed code, and all code running as a startup application on a CompactRIO controller is by default locked and cannot be opened. Unlike other off-the-shelf controllers or some PLCs for which the raw source code is stored on the controller and protected only by a password, CompactRIO systems do not require the raw source code to be stored on the controller.

Code running on the real-time processor is compiled into an executable and cannot be “decompiled” back to LabVIEW code. Likewise, code running on the FPGA has been compiled into a bitfile and cannot be decompiled back to LabVIEW code. To aid in future debugging and maintenance, you can store the LabVIEW project on the controller or call raw VIs from running code, but by default any code deployed to a real-time controller is protected to prevent copying or modifying the algorithms.

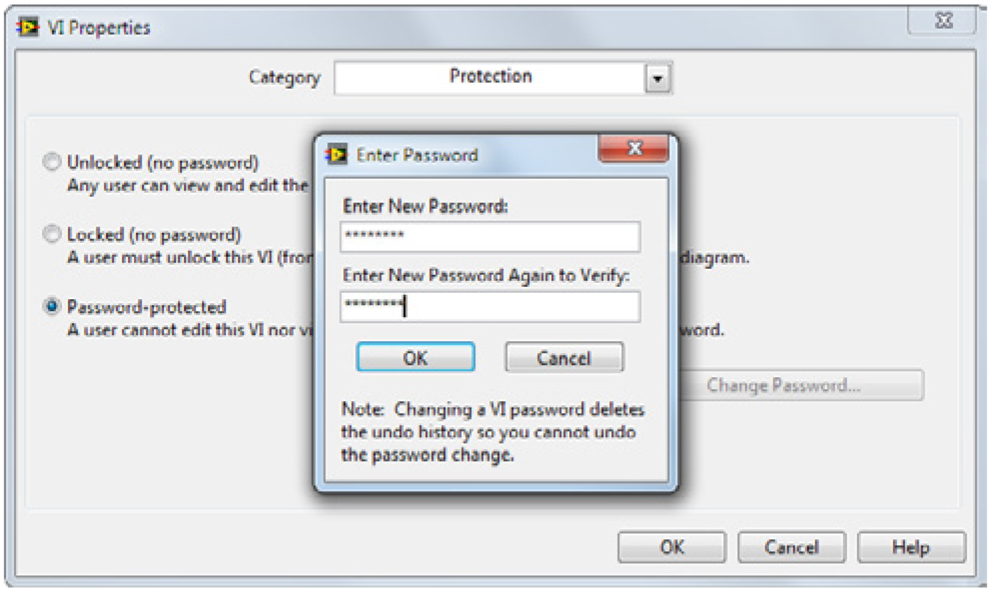
Protect Individual VIs

Sometimes you want to provide the raw LabVIEW code to enable end customers to perform customization or maintenance but still want to protect specific algorithms. LabVIEW offers a few ways to provide usable subVIs while protecting the IP in those VIs.

Method 1: Password Protecting Your LabVIEW Code

Password protecting a VI adds functionality that requires users to enter a password if they want to edit or view the block diagram of a particular VI. Because of this, you can give a VI to someone else and protect your source code. Password protecting a LabVIEW VI prohibits others from editing the VI or viewing its block diagram without the password. However, if the password is lost, you cannot unlock a VI. Therefore, you should strongly consider keeping a backup of your files stored without passwords in another secure location.

To password protect a VI, go to **File»VI Properties.** Choose Protection for the category. This gives you three options: unlocked (the default state of a VI), locked (no password), and password-protected. When you click on password-protected, a window appears for you to enter your password. The password takes effect the next time you launch LabVIEW.


VI Properties

Category: Protection

Unlocked (no password)
Any user can view and edit the

Locked (no password)
A user must unlock this VI (fro ... diagram.

Password-protected
A user cannot edit this VI nor vi ... word.

Enter Password

Enter New Password:
****

Enter New Password Again to Verify:
****

Note: Changing a VI password deletes the undo history so you cannot undo the password change.

Figure 11.28. Password Protecting LabVIEW Code

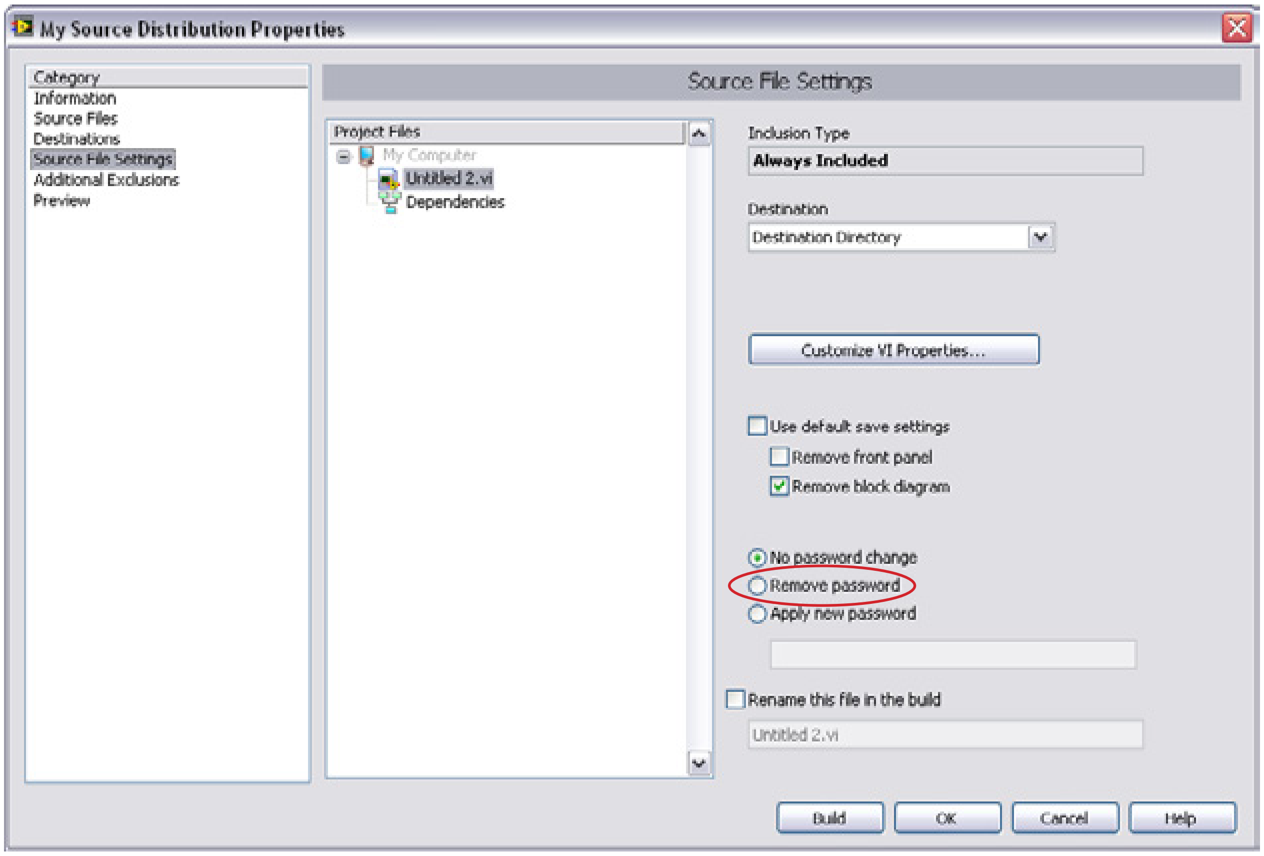
The LabVIEW password mechanism is quite difficult to defeat, but no password algorithm is 100 percent secure from attack. If you need total assurance that someone cannot gain access to your source code, you should consider removing the block diagram.

Method 2: Removing the Block Diagram

To go beyond the protection offered by password protecting a VI and to guarantee that a VI cannot be modified or opened, you can remove the block diagram completely. Much like an executable, the code you distributed no longer contains the original editable code. Do not forget to make a backup of your files if you use this technique because the block diagram cannot be recreated. Removing the block diagram is an option you can select when creating a source distribution. A source distribution is a collection of files that you can package and send to other developers to use in LabVIEW. You can configure settings for specified VIs to add passwords, remove block diagrams, or apply other settings.

Complete the following steps to build a source distribution.

- In the LabVIEW project, right-click **Build Specifications** and select **New»Source Distribution** from the shortcut menu to display the Source Distribution Properties dialog box. Add your VI(s) to the distribution.
- On the **Source File Settings** page of the **Source Distribution Properties** dialog box, remove the checkmark from the **Use default save settings** checkbox and place a checkmark in the **Remove block diagram** checkbox to ensure that LabVIEW removes the block diagram.
- Build the source distribution to create a copy of the VI without its block diagram.



Inclusion Type: Always Included

Destination: Destination Directory

[ ] Use default save settings

[ ] Remove front panel

[✓] Remove block diagram

(●) No password change

( ) Remove password

( ) Apply new password

[ ] Rename this file in the build: Untitled 2.vi

Figure 11.29. Removing the Block Diagram From LabVIEW VIs

Note: If you save VIs without block diagrams, do not overwrite the original versions of the VIs. Save the VIs in different directories or use different names.

Lock Code to Hardware to Prevent IP Replication

Some OEMs and machine builders also want to protect their IP by locking the deployed code to a specific system. To make system replication easy, by default the deployed code on a CompactRIO controller is not locked to hardware and can be moved and executed on another controller. For designers who want to prevent customers or competitors from replicating their systems, one effective way to protect application code with CompactRIO is by locking your code to specific pieces of hardware in your system. This ensures that customers cannot take the code off a system they have purchased from you and run the application on a different set of CompactRIO hardware. You can lock the application code to a variety of hardware components in a CompactRIO system including the following:

- The MAC address of a real-time controller
- The serial number of a real-time controller
- The serial number of the CompactRIO backplane
- The serial number of individual modules
- Third-party serial dongle

You can use the following steps as guidelines to programmatically lock any application to any of the above mentioned hardware parameters and thus prevent users from replicating application code:

1. Obtain the hardware information for the device. Refer to the following procedures for more guidance on programmatically obtaining this information.
2. Compare the values obtained to a predetermined set of values that the application code is designed for using the **Equal?** function from the **Comparison** palette.
3. Wire the results of the comparison to the **selector** input of a **Case structure.**
4. Place the application code in the **true** case and leave the **false** case blank.

Performing these steps ensures that the application is not replicated or usable on any other CompactRIO hardware.

License Key

Adding licensing to a LabVIEW Real-Time application can protect a deployed application from being copied and run on another similar or identical set of hardware without obtaining a license from the vendor or distributor of the application. Most modern-day applications running on desktop computers are protected by a license key that is necessary to either install the application or run it in its normal operational mode. Many vendors use license keys to determine if an application runs in demo mode or is fully functional. License keys may also be used to differentiate between versions of an application or to enable/disable specific features.

You can add this behavior to a LabVIEW Real-Time application using the reference design and example code developed by NI Systems Engineering. You can download this code from the NI Developer Zone document Reference Design for Adding Licensing to LabVIEW Real-Time Applications. The main modification is how you create a unique system ID for a specific hardware target. In the case of CompactRIO, you use the controller, backplane, and module serial numbers. For other targets, you may use the serial number or Ethernet MAC address of a given target to create a unique system ID.

Choosing a License Model

The license model defines how the license key is generated based on particular characteristics of the target hardware. One simple example of a license model is to base the license key on the serial number of the controller. In this case, the application runs if the controller has the serial number matching the license key provided. If the controller serial number does not match the license key, the application does not run.

If the hardware target includes additional components such as a CompactRIO backplane and CompactRIO modules, you may choose to lock the application not only to the controller but also to these additional system components. In this case, you can link the license key to the serial numbers of all of these hardware target components. All components with the correct serial numbers must be in place to run the application.

The unique characteristic of the hardware target (for example, a serial number) is called the system ID for the purpose of this guide.

One example of a more complex license model is to base the license key on the serial numbers of multiple system components but require only some of these components to be present when running the licensed application. This allows the application user to replace some of the system components, in case a repair is required, without needing to acquire a new application license key. The number of components linked to the license key that must be present to run the application is defined by the developer as part of the license model.

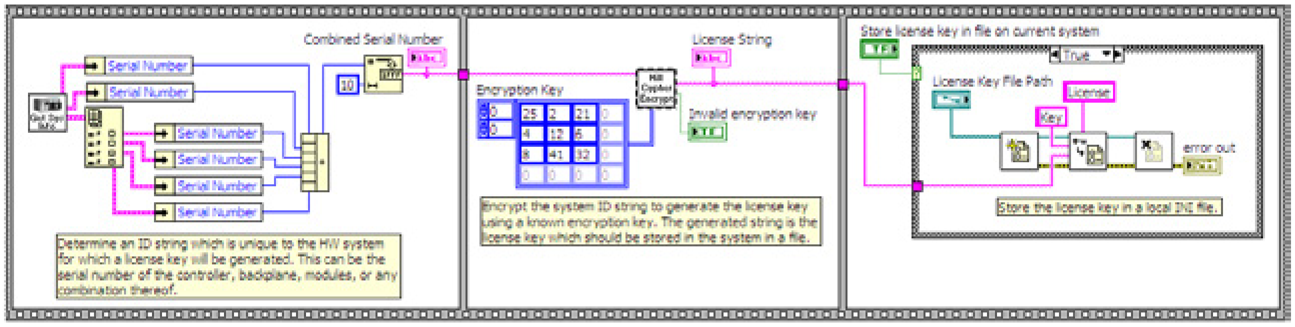
Application Licensing Process

Adding licensing (creating and using a license key) to a LabVIEW Real-Time application consists of the following steps:

1. Create a unique system ID for each deployed hardware target
2. Create a license key based on the system ID
3. Store the license key on the hardware target
4. Verify the license key during application startup

Create a Unique System ID for Each Deployed Hardware Target

To create a license key for a specific hardware target, you must first create a unique system ID. The system ID is a number or string that uniquely identifies a specific hardware target based on the physical hardware itself. The licensed application is locked to the hardware target using the license key, which is linked to the system ID. The system ID can be a number such as the serial number of the target or a value that is a combination of each of the system component's serial numbers. Another source for a system ID can be the Media Access Control (MAC) address stored in every Ethernet interface chipset. The example uses the Reference Library for Reading CompactRIO System Configuration Information to retrieve these different pieces of information from a CompactRIO system.



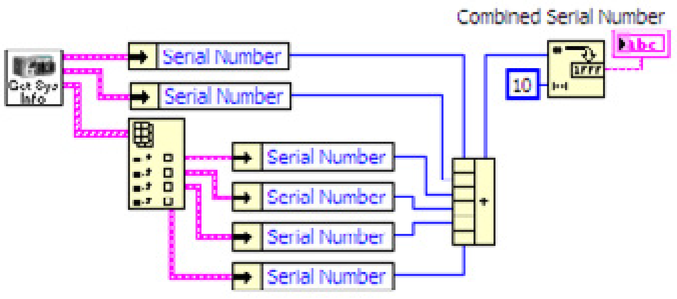
Determine an ID string which is unique to the HW system for which a license key will be generated. This can be the serial number of the controller, backplane, modules, or any combination thereof.

Encrypt the system ID string to generate the license key using a known encryption key. The generated string is the license key which should be stored in the system in a file.

Store the license key in a local INI file.

Figure 11.30. Block Diagram Showing the Process of Creating the License Key

Figure 11.31 shows one possible example of generating a system ID for a 4-slot CompactRIO system using these VIs. The serial numbers for all six system components are added together. While this is not a truly unique value for a given CompactRIO system, it is unlikely that after replacing one or more system components, the sum of all the serial numbers will be the same as when the license key was generated.



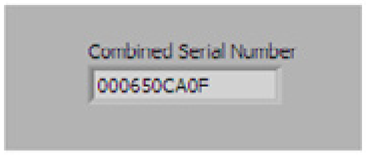

Combined Serial Number
000650CA0F

Figure 11.31. Generating a System ID Using Serial Numbers

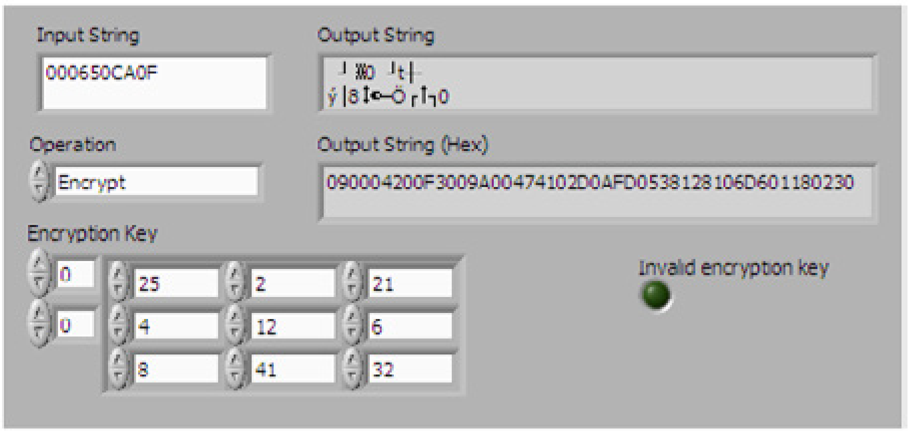
Create a License Key Based on the System ID

Once you create a system ID, you derive a license key from the system ID. The license key is a unique string that is linked to the system ID using an encryption algorithm. The goal is to create a string that can be verified against the system ID but cannot be easily generated by the user for a different hardware target to disallow using the application on a different set of hardware.

Encryption

The encryption used in the reference example is a version of the Hill Cipher, which uses a simple matrix multiplication and a license key matrix. The matrix multiplication converts the input string into a different output string that cannot be easily decrypted without knowing the key matrix. The Hill Cipher is not considered to be a strong encryption algorithm and is only one possible option you can use to generate a license key. You can easily replace the encryption subVI in the reference example with your own encryption tool.

The encryption VI provided with the reference example converts the system ID using the Hill Cipher and a 3x3 key matrix (Encryption Key). The key matrix is the key to unlocking the license key; therefore, you should change the key matrix value before using the reference example in a real-world application. Not all 3x3 matrices are valid for use as a key matrix. The Hill Cipher Encryption VI tells you if your chosen matrix is invalid. If it is invalid, try other values for your key matrix.


Input String
000650CA0F

Output String
[illegible]

Operation
Encrypt

Output String (Hex)
090004200F3009A00474102D0AFD0538128106D601180230

Encryption Key
0
0

25	2	21
4	12	6
8	41	32

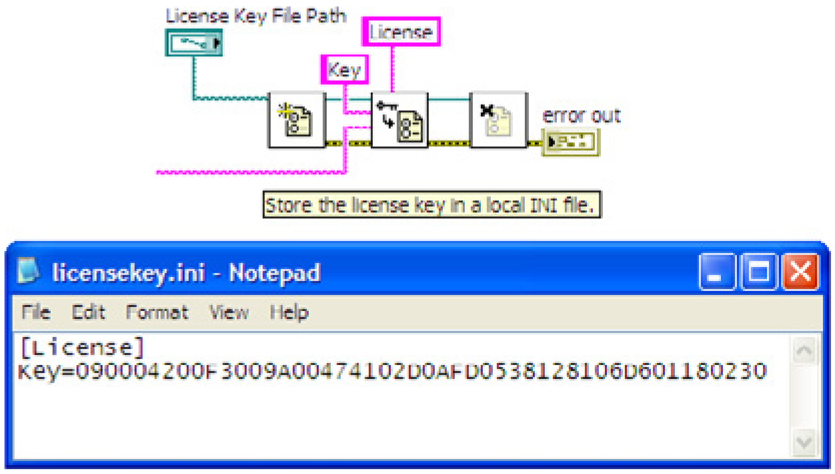
Invalid encryption key

Figure 11.32. Encryption of the System ID Into the License Key

The encryption VI provides two versions of the license key. The default algorithm returns a stream of bytes that can have any value between 0 and 255; therefore, these bytes may not represent printable characters and may not be easily stored in a text file or passed along using other mechanisms such as email. To simplify the process of storing and transferring the license key, the VI provides a hexadecimal version of the encrypted string made up of the ASCII representation of the hexadecimal value of each of the byte values in the original string. This string is stored in a file as the license key.

Store the License Key on the Hardware Target

The reference example stores the license key in a simple INI file on the CompactRIO controller hard drive.


License Key File Path
License
Key
error out
Store the license key in a local INI file.
licensekey.ini - Notepad
File Edit Format View Help

```
[License]
Key=090004200F3009A00474102D0AFD0538128106D601180230
```

Figure 11.33. Storing the License Key in an INI File on the CompactRIO Controller

If the license key and file are generated away from the actual CompactRIO system, then you must copy the license file to the CompactRIO system when you deploy the application to the controller.

Verify the License Key During Application Startup

When the deployed application is starting up, it needs to verify the license key and, based on the results of the verification process, adjust its behavior according to the license model. If the license key is verified correctly, the application runs normally, but if the key is not verified, it may not run at all or run in an evaluation mode.

The reference example provides a basic verification VI that is added before the actual application.

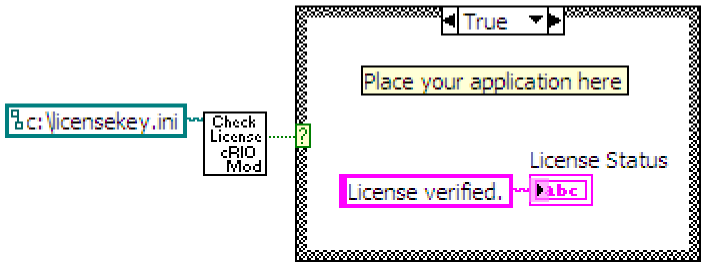
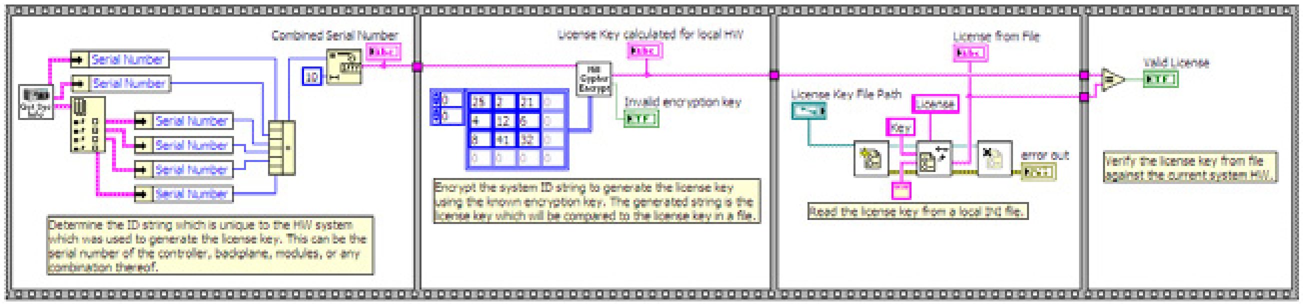


Figure 11.34. Adding the License Key Verification to an Application

You can choose from two different methods to verify a license key. The first and preferred method is to re-create the license key on the target system (described in this section). The second method, which consists of decrypting the license key, is described in the NI Developer Zone white paper Reference Design for Adding Licensing to LabVIEW Real-Time Applications under the section titled "Enabling Features based on the License Key".

The more basic and more secure method to verify the license key is to run the same algorithm you use to create the license key on the target system and then compare the new license key with the license key stored in the license file. If the two match, then the license key is verified and the application may run.



Determine the ID string which is unique to the HW system which was used to generate the license key. This can be the serial number of the controller, backplane, modules, or any combination thereof.

Encrypt the system ID string to generate the license key using the known encryption key. The generated string is the license key which will be compared to the license key in a file.

Read the license key from a local INI file.

Verify the license key from file against the current system HW.

Figure 11.35. Block Diagram to Verify the License Key Stored in the File on the System

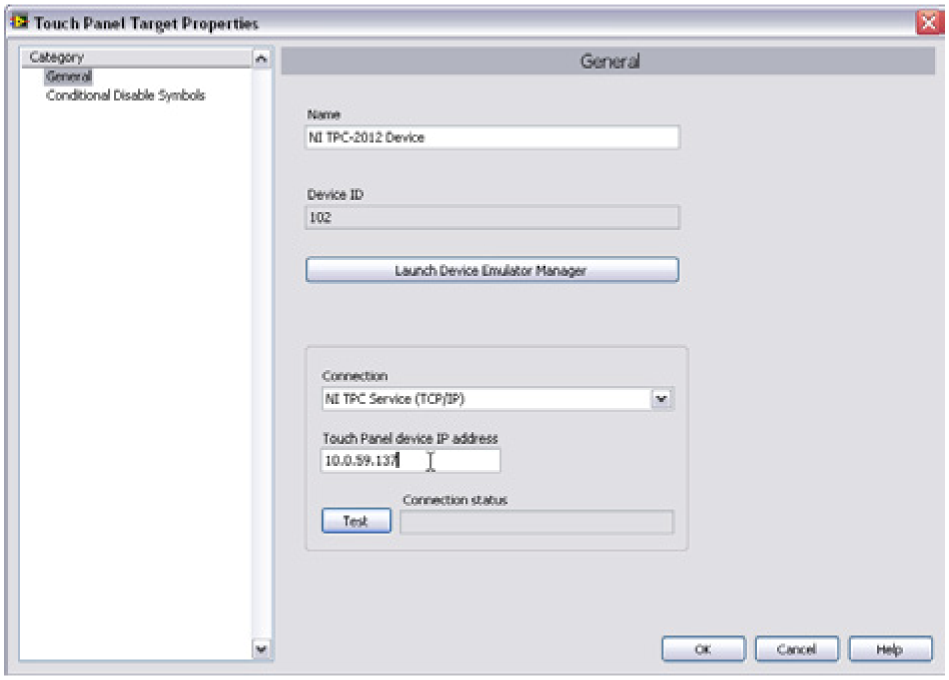
Figure 11.35 shows that this process is almost identical to the process of generating the license key. Instead of writing the license file, however, the license file is read and compared to the newly generated license key.

This method of verifying the license key works well if you do not need any partial information from the license key such as information about enabling or disabling individual features or the individual serial numbers of system components. For licensing models that require more detailed information about the license key, the key itself must be decrypted. For more information on this type of licensing, see the NI Developer Zone document Reference Design for Adding Licensing to LabVIEW Real-Time Applications.

Deploying Applications to a Touch Panel

Configure the Connection to the Touch Panel

Although you can manually copy built applications to a touch panel device, you should use Ethernet and allow the LabVIEW project to automatically download the application. NI touch panels are all shipped with a utility called the NI TPC Service that allows the LabVIEW project to directly download code over Ethernet. To configure the connection, right-click on the touch panel target in the LabVIEW project and select Properties. In the General category, choose the NI TPC Service connection and enter the IP address of the touch panel. Test the connection to make sure the service is running.


Touch Panel Target Properties
Category
General
Conditional Disable Symbols
General
Launch Device Emulator Manager
Test
Connection status
OK
Cancel
Help

Name: NI TPC-2012 Device

Device ID: 102

Connection: NI TPC Service (TCP/IP)

Touch Panel device IP address: 10.0.59.137

Figure 11.36. Connect to a touch panel through Ethernet using the NI TPC Service.

You can find the IP address of the touch panel by going to the command prompt on the TPC and typing ipconfig. To get to the command prompt, go to the Start menu and select Run… In the pop-up window, enter cmd.

Deploy a LabVIEW VI to Volatile or Nonvolatile Memory

The steps to deploy an application to a Windows XP Embedded touch panel and to a Windows CE touch panel are nearly identical. The only difference is on an XP Embedded touch panel, you can deploy an application to only the nonvolatile memory, and, on a Windows CE touch panel, you can deploy to volatile or nonvolatile memory, depending on the destination directory you select. To run a deployed VI in either volatile or nonvolatile memory on a touch panel, you must first create an executable.

Building an Executable From a VI for an XP Embedded Touch Panel

The LabVIEW project provides the ability to build an executable touch panel application from a VI. To do this, you create a build specification under the touch panel target in the LabVIEW Project Explorer. By right-clicking on Build Specifications, you can select the option of creating a Touch Panel Application, Source Distribution, Zip File, and so on.

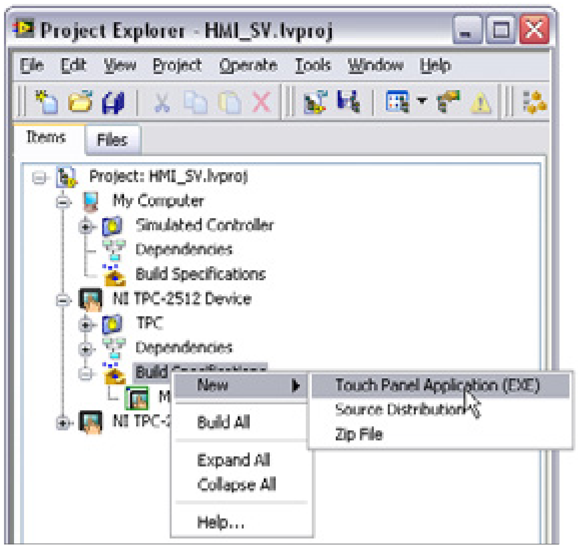
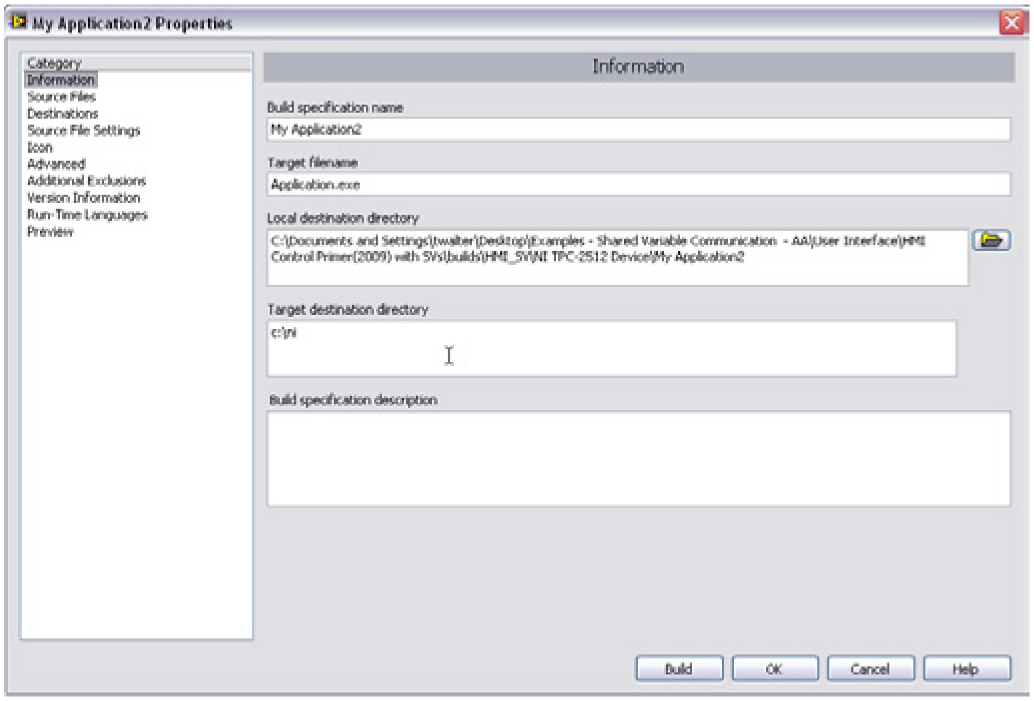


Figure 11.37. Create a touch panel application using the LabVIEW project.

After selecting the Touch Panel Application, you are presented with a dialog box. The two most commonly used categories when building a touch panel application are Information and Source Files. The other categories are rarely changed when building touch panel applications.

The Information category contains the build specification name, executable filename, and destination directory for both the touch panel target and host PC. You can change the build specification name and local destination directory to match your nomenclature and file organization. You normally do not need to change the target filename or target destination directory.



Build specification name

My Application2

Target filename

Application.exe

Local destination directory

C:\Documents and Settings\twalter\Desktop\Examples - Shared Variable Communication - AA\User Interface\HMI Control Primer(2009) with SVs\builds\HMI_SV\NI TPC-2512 Device\My Application2

Target destination directory

c:\ni

Build specification description

Figure 11.38. The Information Category in the Touch Panel Application Properties

You use the Source Files category to set the startup VIs and obtain additional VIs or support files. You need to select the top-level VI from your Project Files and set it as a Startup VI. For most applications, a single VI is chosen to be a Startup VI. You do not need to include lvlib or set subVIs as Startup VIs or Always Included unless they are called dynamically in your application.

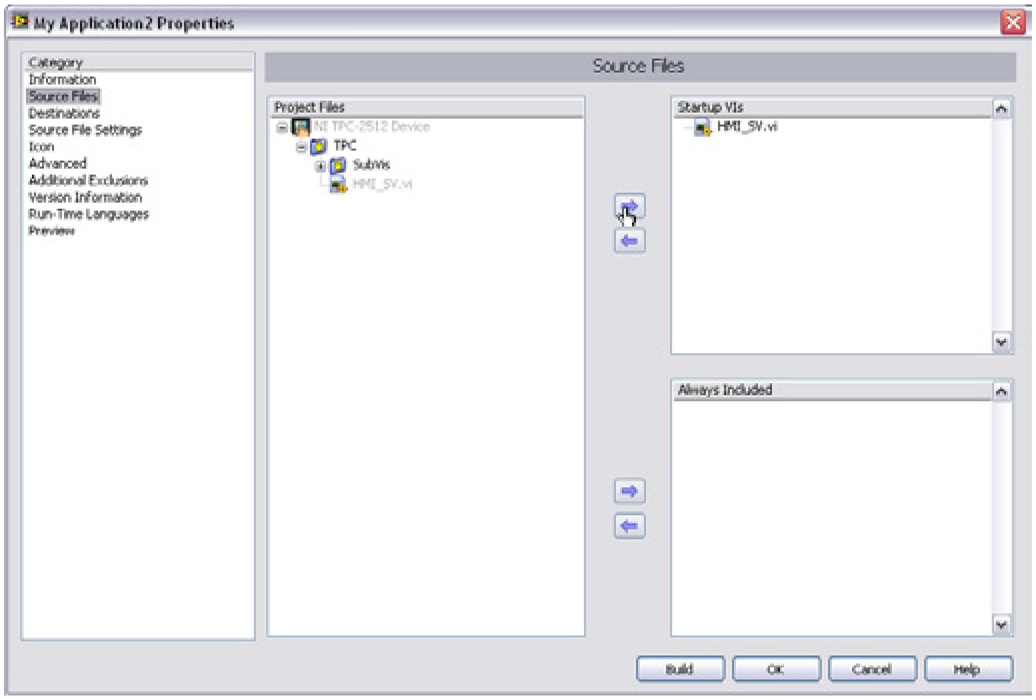


Figure 11.39. Source Files Category in the Touch Panel Application Properties
(In this example, the HMI_SV.vi was selected to be a Startup VI.)

After you have entered all of the information on the required category tabs, you can click OK to save the build specification or you can directly build the application by clicking the Build button. You can also right-click on a saved build specification and select Build to build the application.

When you build the application, an executable is created and saved on the hard drive of your development machine in the local destination directory.

Building an Executable From a VI for a Windows CE Touch Panel

The LabVIEW project provides the ability to build an executable touch panel application from a VI. To build this application, you create a build specification under the touch panel target in the LabVIEW Project Explorer. By right-clicking on Build Specifications, you can select the option of creating a Touch Panel Application, Source Distribution, Zip File, and so on.

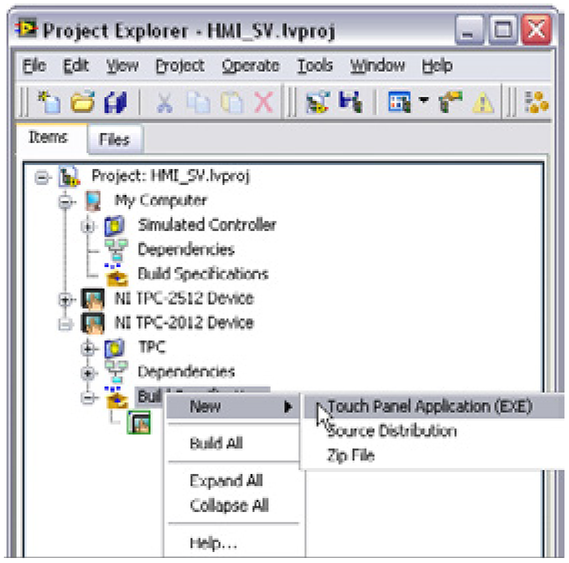
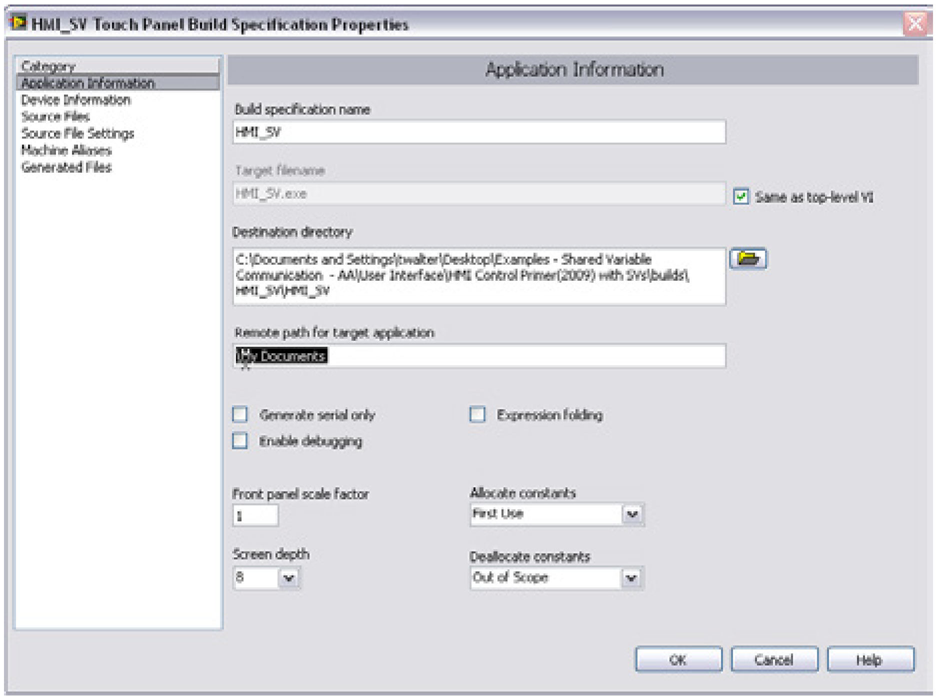


Figure 11.40. Creating a Touch Panel Application Using the LabVIEW Project

After selecting Touch Panel Application, you see a dialog box with the three main categories that are most commonly used when building a touch panel application for a Windows CE target: Application Information, Source Files, and Machine Aliases. The other categories are rarely changed when building Windows CE touch panel applications.

The Application Information category contains the build specification name, executable filename, and destination directory for both the touch panel target and host PC. You can change the build specification name and local destination directory to match your nomenclature and file organization. You normally do not need to change the target filename. The target destination determines if the deployed executable runs in volatile or nonvolatile memory. On a Windows CE device

- \My Documents folder is volatile memory. If you deploy the executable to this memory location, it does not persist through power cycles.
- \HardDisk is nonvolatile memory. If you want your application to remain on the Windows CE device after a power cycle, you should set your remote path for target application to a directory on the \HardDisk such as \HardDisk\Documents and Settings.


HMI_SV Touch Panel Build Specification Properties

Category
- Application Information
- Device Information
- Source Files
- Source File Settings
- Machine Aliases
- Generated Files

Application Information

Build specification name: HMI_SV

Target filename: HMI_SV.exe ☑ Same as top-level VI

Destination directory: C:\Documents and Settings\twalter\Desktop\Examples - Shared Variable Communication - AA\User Interface\HMI Control Primer(2009) with SVs\builds\HMI_SV\HMI_SV

Remote path for target application: My Documents

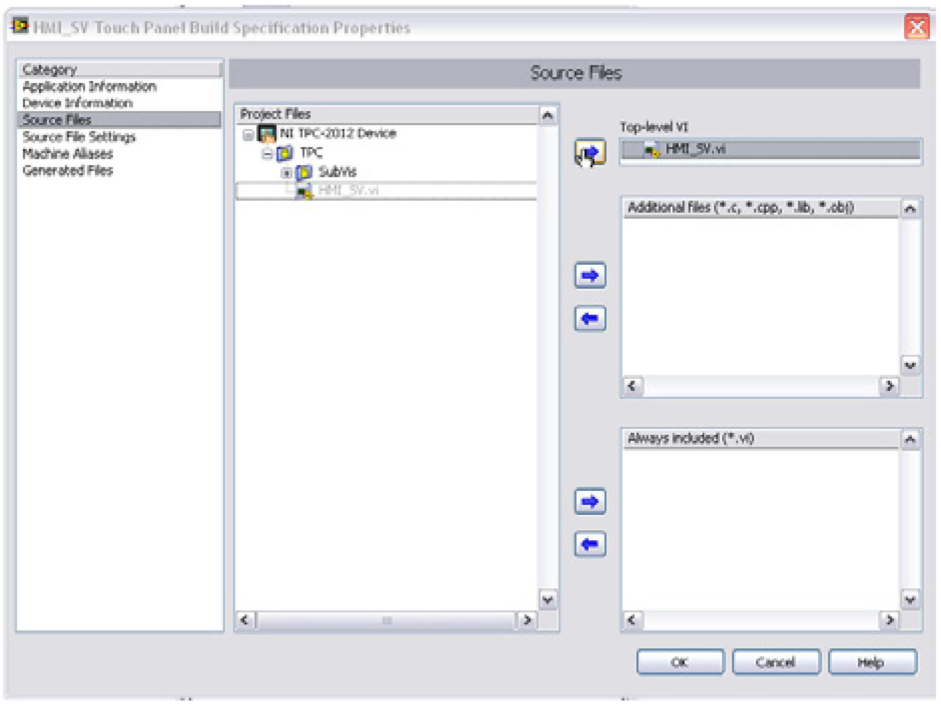
☐ Generate serial only ☐ Expression folding
☐ Enable debugging

Front panel scale factor: 1 Allocate constants: First Use

Screen depth: 8 Deallocate constants: Out of Scope

Figure 11.41. The Information Category in the Touch Panel Application Properties

Use the Source Files category to set the startup VI and obtain additional VIs or support files. You need to select the top-level VI from your Project File. The top-level VI is the startup VI. For Windows CE touch panel applications, you can select only a single VI to be the top-level VI. You do not need to include lvlib or subVIs as Always Included.


HMI_SV Touch Panel Build Specification Properties

Category
- Application Information
- Device Information
- Source Files
- Source File Settings
- Machine Aliases
- Generated Files

Source Files

Project Files
- NI TPC-2012 Device
  - TPC
    - SubVIs
    - HMI_SV.vi

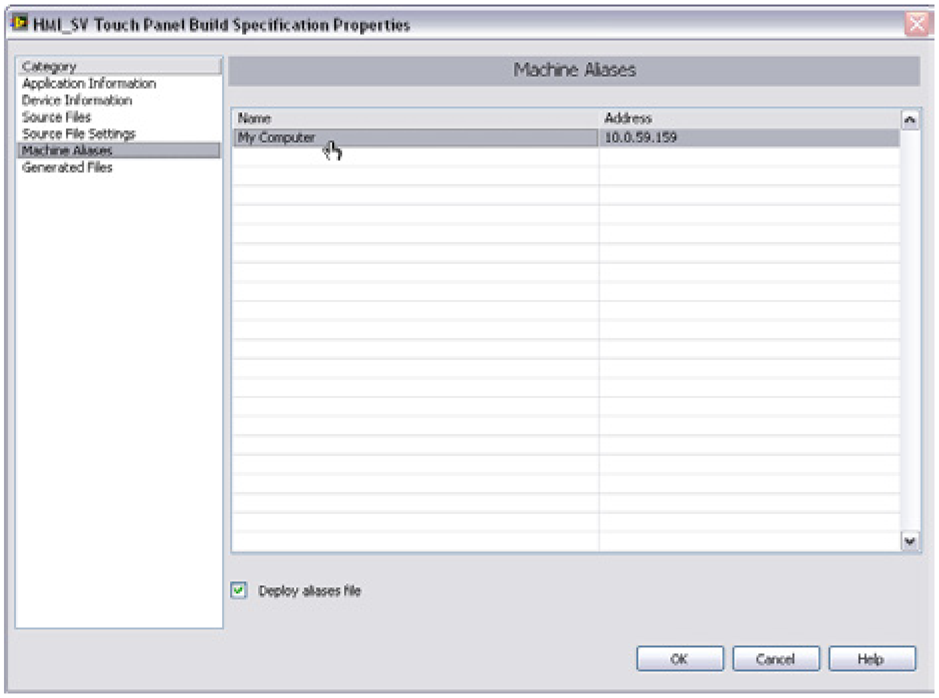
Top-level VI: HMI_SV.vi

Additional files (*.c, *.cpp, *.lib, *.obj)

Always included (*.vi)

Figure 11.42. The Source Files Category in the Touch Panel Application Properties (In this example, the HMI_SV.vi was selected to be the top-level VI.)

The Machine Aliases category is used to deploy an alias file. This is required if you are using network-published shared variables for communication to any devices. Be sure to check the Deploy alias file checkbox. The alias list should include your network-published shared variable servers and their IP addresses (normally CompactRIO or Windows PCs). You can find more information on alias files and application deployment using network-published shared variables in the section titled Deploying Applications That Use Network-Published Shared Variables.



Name	Address
My Computer	10.0.59.159

☑ Deploy aliases file

Figure 11.43. The Machine Aliases Category in the Touch Panel Application Properties (Be sure to check the deploy aliases file checkbox if you are using network-published shared variables.)

After you have entered all of your information on the required category tabs, you can click OK to save the build specification or you can directly build the application by clicking the Build button. You can also right-click on a saved build specification and select Build to build the application.

When you build the application, an executable is created and saved on the hard drive of your development machine in the local destination directory.

Deploy an Executable Touch Panel Application to a Windows CE or XP Embedded Target

After configuring and building your executable, you now need to copy the executable and supporting files to the memory on the touch panel. To copy the files, right-click on the Touch Panel Application and select Deploy. Behind the scenes, LabVIEW copies the executable files to the memory on the touch panel. If you rebuild an application, you must redeploy the touch panel application for the changes to take effect on the touch panel target.

The Run Button

If you click the Run button on a VI targeted to a touch panel target, LabVIEW guides you through creating a build specification (if one does not exist) and deploys the code to the touch panel target.

Setting an Executable Touch Panel Application to Run on Startup

After you have deployed an application to the touch panel, you can set the executable so it automatically starts up as soon as the touch panel boots. Because you are running on a Windows system, you do this using standard Windows tools. In Windows XP Embedded, you should copy the executable and paste a shortcut into the Startup directory on the Start Menu. On Windows CE, you need to go to the STARTUP directory on the hard disk and modify the startup.ini file to list the path to the file (\HardDisk\Documents and Settings\HMI_SV.exe). You can alternatively use the

Misc tab in the Configuration Utility (**Start»Programs»Utilities»Configuration Utilities**) to configure a program to start up on boot. This utility modifies the startup.ini file for you.

Porting to Other Platforms

This guide has focused on architectures for building embedded control systems using CompactRIO systems. The same basic techniques and structures also work on other NI control platforms including PXI and NI Single-Board RIO. Because of this, you can reuse your algorithms and your architecture for other projects that require different hardware or easily move your application between platforms. However, CompactRIO has several features to ease learning and speed development that are not available on all targets. This section covers the topics you need to consider when moving between platforms and shows you how to port an application to NI Single-Board RIO.

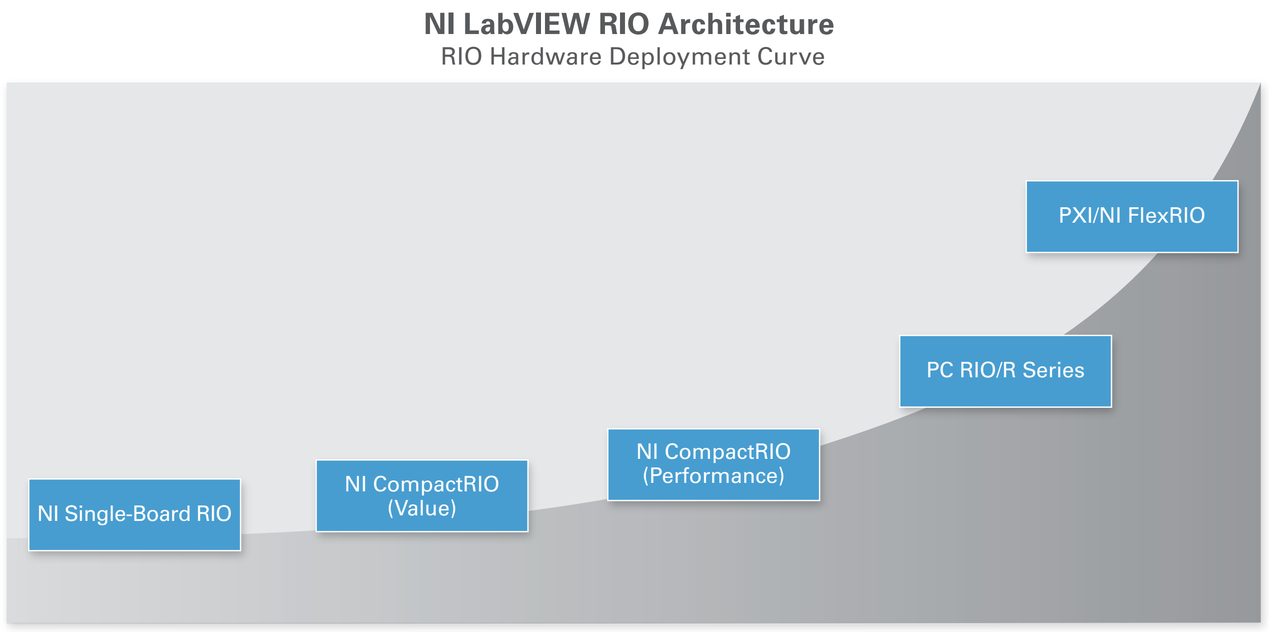

NI LabVIEW RIO Architecture
RIO Hardware Deployment Curve
PXI/NI FlexRIO
PC RIO/R Series
NI CompactRIO (Performance)
NI CompactRIO (Value)
NI Single-Board RIO

Figure 11.44. With LabVIEW, you can use the same architecture for applications ranging from CompactRIO to high-performance PXI to board-level NI Single-Board RIO.

LabVIEW Code Portability

LabVIEW is a cross-platform programming language capable of compiling for multiple processor architectures and OSs. In most cases, algorithms written in LabVIEW are portable among all LabVIEW targets. In fact, you can even take LabVIEW code and compile it for any arbitrary 32-bit processor to port your LabVIEW code to custom hardware. When porting code between platforms, the most commonly needed changes are related to the physical I/O changes of the hardware.

When porting code between CompactRIO targets, all I/O is directly compatible because C Series modules are supported on all CompactRIO targets. If you need to port an application to NI Single-Board RIO, all C Series modules are supported, but, depending on your application, you may need to adjust the software I/O interface.

NI Single-Board RIO

NI Single-Board RIO is a board-only version of CompactRIO designed for applications requiring a bare board form factor. While it is physically a different design, NI Single-Board RIO uses the processor and FPGA, and most models accept up to three C Series modules. NI Single-Board RIO differs from CompactRIO because it includes I/O built directly into the board. NI offers two families of NI-Single-Board RIO products:

Digital I/O With RIO Mezzanine Card Connector

The smallest option for NI Single-Board RIO combines the highest performance real-time processor with a Xilinx Spartan-6 FPGA and built-in peripherals such as USB, RS232, CAN, and Ethernet. In addition to the peripherals, the system includes 96 FPGA digital I/O lines that are accessed through the RIO Mezzanine Card (RMC) connector, which is a high-density, high-bandwidth connector that allows for direct access to the FPGA and processor. With this type of NI Single-Board RIO, you can create a customized daughter card designed specifically for your application that accesses the digital I/O lines and processor I/O including CAN and USB. This NI Single-Board RIO family currently does not support C Series connectivity.

Digital I/O Only or Digital and Analog I/O With Direct C Series Connectivity

NI also offers NI Single-Board RIO devices with both built-in digital and analog I/O on a single board. All I/O is connected directly to the FPGA, providing low-level customization of timing and I/O signal processing. These devices feature 110 3.3 V bidirectional digital I/O lines and up to 32 analog inputs, 4 analog outputs, and 32 24 V digital input and output lines, depending on the model used. They can directly connect up to three C Series I/O and communication modules for further I/O expansion and flexibility.

LabVIEW FPGA Programming

Not all NI Single-Board RIO products currently support Scan Mode. Specifically, the NI Single-Board RIO products with a 1 million gate FPGA (sbRIO-9601, sbRIO-9611, sbRIO-9631, and sbRIO-9641) are not supported, in addition to the NI Single-Board RIO products with the RMC connector (sbRIO-9605 and sbRIO-9606). Instead of using Scan Mode to read I/O, you need to write a LabVIEW program to read the I/O from the FPGA and insert it into an I/O memory table. This section examines an effective FPGA architecture for single-point I/O communication similar to Scan Mode and shows how to convert an application using Scan Mode.

Built-In I/O and I/O Modules

Depending on your application I/O requirements, you may be able to create your entire application to use only the NI Single-Board RIO onboard I/O, or you may need to add modules. When possible, design your application to use the I/O modules available onboard NI Single-Board RIO. The I/O available on NI Single-Board RIO with direct C Series connectivity and the module equivalents are listed below:

- 110 general-purpose, 3.3 V (5 V tolerant, TTL compatible) digital I/O (no module equivalent)
- 32 single-ended/16 differential channels, 16-bit analog input, 250 kS/s aggregate (NI 9205)
- 4-channel, 16-bit analog output; 100 kS/s simultaneous (NI 9263)
- 32-channel, 24 V sinking digital input (NI 9425)
- 32-channel, 24 V sourcing digital output (NI 9476)

This NI Single-Board RIO family accepts up to three additional C Series modules. Applications that need more than three additional I/O modules are not good candidates for NI Single-Board RIO, and you should consider CompactRIO integrated systems as a deployment target.

FPGA Size

The largest FPGA available on NI Single-Board RIO is the Spartan-6 LX45. CompactRIO targets offer versions using both the Virtex-5 FPGAs and the Spartan-6 FPGAs as large as the LX150. To test if code fits on hardware you do not own, you can add a target to your LabVIEW project and, as you develop your FPGA application, you can periodically benchmark the application by compiling the FPGA code for a simulated RIO target. This gives you a good understanding of how much of your FPGA application will fit on the Spartan-6 LX45.

Port CompactRIO Applications to NI Single-Board RIO or R Series Devices

Follow these four main steps to port a CompactRIO application to NI Single-Board RIO or PXI/PCI R Series FPGA I/O devices.

5. Build an NI Single-Board RIO or R Series project with equivalent I/O channels.
6. If using the NI RIO Scan Interface, build a LabVIEW FPGA-based scan API if porting to unsupported NI Single-Board RIO devices or to PXI/PCI R Series FPGA I/O Devices
    - Build LabVIEW FPGA I/O scan (analog in, analog out, digital I/O, specialty digital I/O).
    - Convert I/O variable aliases to single-process shared variables with real-time FIFO enabled.
    - Build a real-time I/O scan with scaling and a shared variable-based current value table.
7. Compile LabVIEW FPGA VI for new target.
8. Test and validate updated real-time and FPGA code.

The first step in porting an application from CompactRIO to NI Single-Board RIO or an R Series FPGA device is finding the equivalent I/O types on your target platform. For I/O that cannot be ported to the onboard I/O built into NI Single-Board RIO or R Series targets, you can add C Series modules. All C Series modules for CompactRIO are compatible with both NI Single-Board RIO and R Series. You must use the NI 9151 R Series expansion chassis to add C Series I/O to an R Series DAQ device.

Step two is necessary only if the application being ported was originally written using the NI RIO Scan Interface, and if porting to unsupported NI Single-Board RIO or PXI/PCI R Series FPGA I/O Devices. If you need to replace the NI RIO Scan Interface portion of an application with an I/O method supported on all RIO targets, an example is included below to guide you through the process.

If the application you are migrating to NI Single-Board RIO or PXI/PCI R Series did not use the RIO Scan Interface, the porting process is nearly complete. Skip step 2 and add your real-time and FPGA source code to your new NI Single-Board RIO project, recompile the FPGA VI, and you are now ready to run and verify application functionality. Because CompactRIO and NI Single-Board RIO are both based on the RIO architecture and reusable modular C Series I/O modules, porting applications between these two targets is simple.

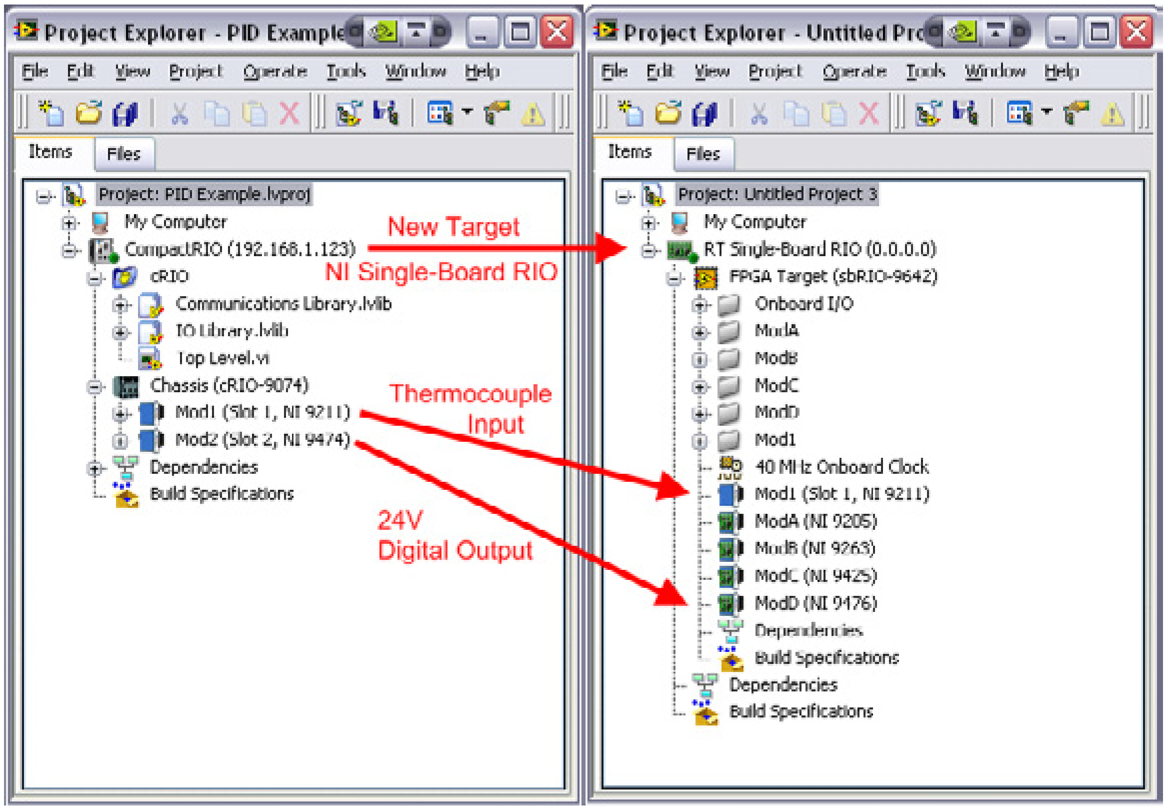


Figure 11.45. The first step in porting an application from CompactRIO to an alternate target is finding replacement I/O on the future target.

Example of Porting a RIO Scan Interface-Based Application to Use LabVIEW FPGA

If you used the RIO Scan Interface in your original application, you might need to create a simplified FPGA version of the Scan Engine. Use these three steps to replace the RIO Scan Interface with a similar FPGA-based scan engine and current value table:

9. Build a LabVIEW FPGA I/O scan engine

10. Replace scan engine I/O variables with single-process shared variables

11. Write FPGA data to a current value table in LabVIEW Real-Time

First, create a LabVIEW FPGA VI that samples and updates all analog input and output channels at the rate specified in your scan engine configuration. You can use IP blocks to recreate specialty digital functionality such as counters, PWM, and quadrature encoders. Next, create an FPGA Scan Loop that synchronizes the I/O updates by updating all outputs and reading the current value of all inputs in sequence.

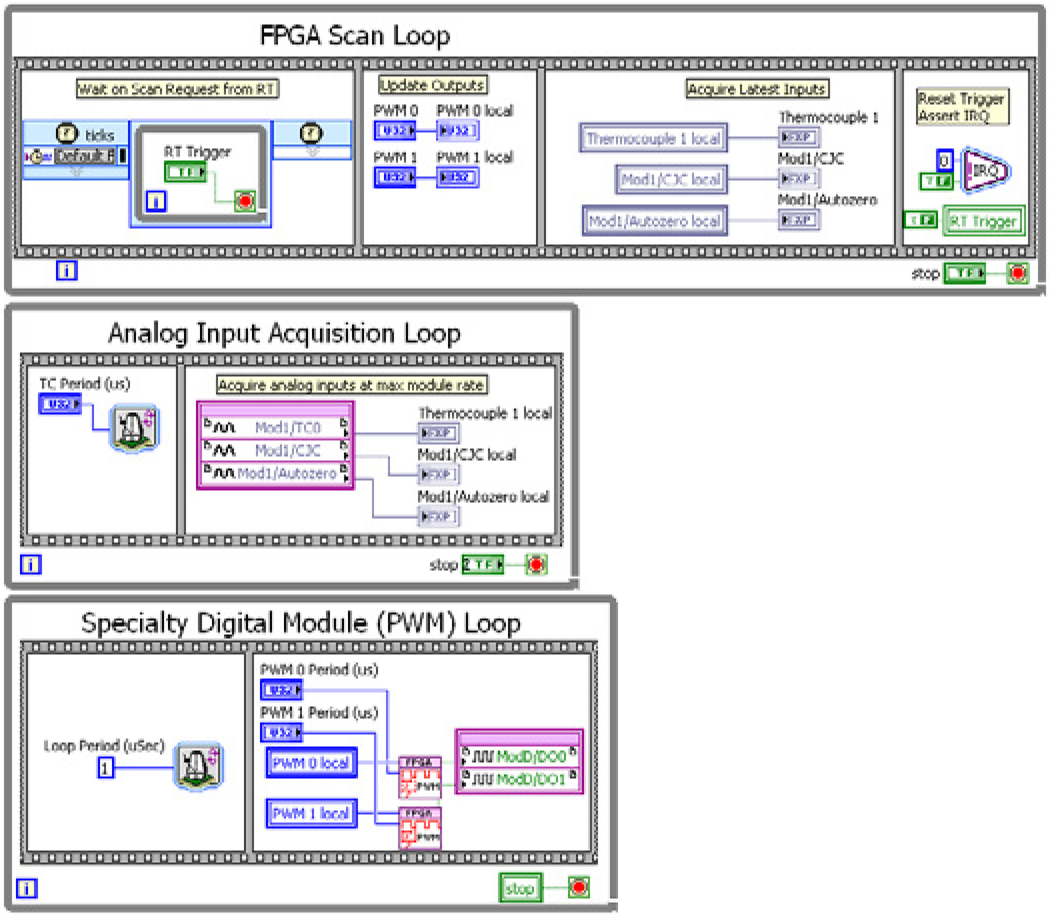
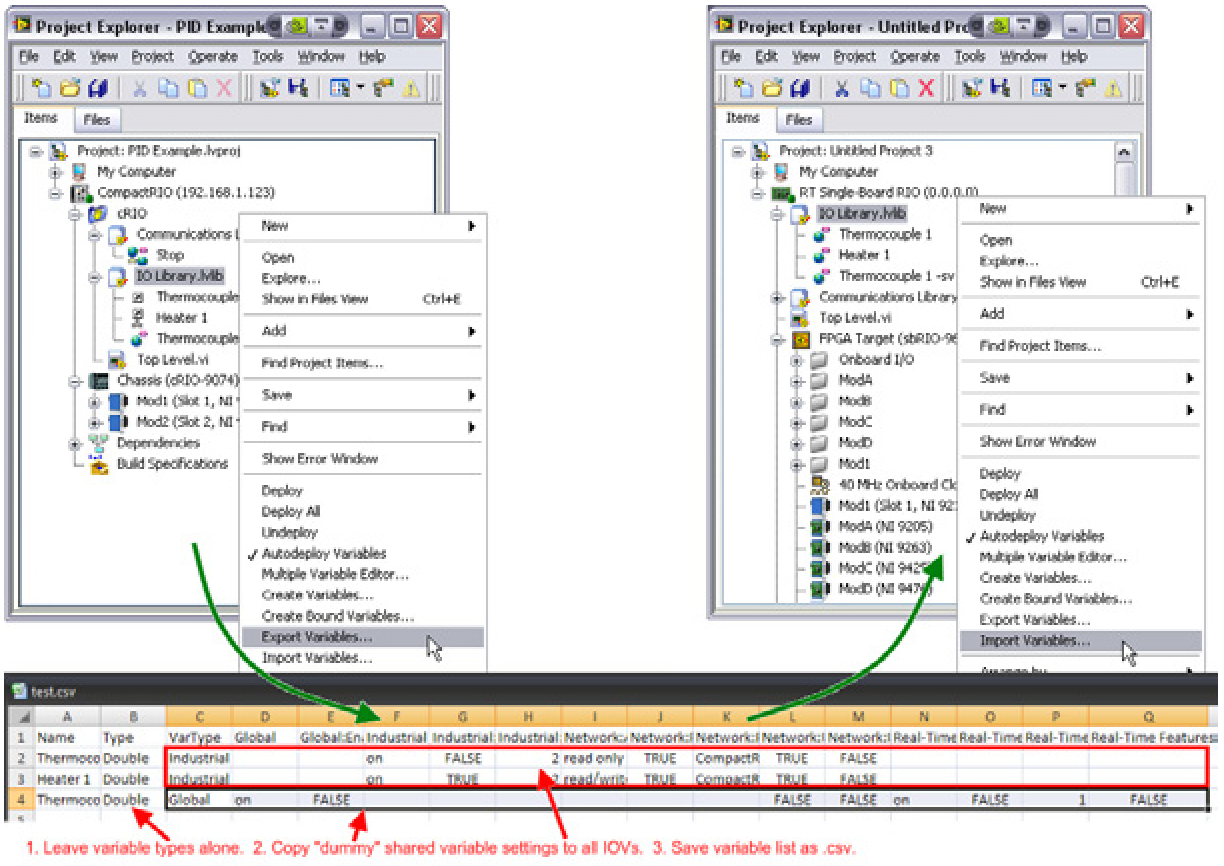


Figure 11.46. Develop a simple FPGA application to act as an FPGA scan engine.

After you have implemented a simple scan engine in the FPGA, you need to port the real-time portion of the application to communicate with the custom FPGA scan engine rather than the scan engine I/O variables. To accomplish this, you need to first convert all I/O variable aliases to single-process shared variables with the real-time FIFO enabled. The main difference between the two variables is while I/O variables are automatically updated by a driver to reflect the state of the input or output channel, single-process shared variables are not updated by a driver. You can change the type by going to the properties page for each I/O variable alias and changing it to single-process.

Tip: If you have numerous variables to convert, you can easily convert a library of I/O variable aliases to shared variables by exporting to a text editor and changing the properties. To make sure you get the properties correct, you should first create one "dummy" single-process shared variable with the single-element real-time FIFO enabled in the library and then export the library to a spreadsheet editor. While in the spreadsheet editor, delete the columns exclusive to I/O variables and copy the data exclusive to the shared variables to the I/O variable rows. Then import the modified library into your new project. The I/O variable aliases are imported as single-process shared variables. Because LabVIEW references shared variables and I/O variable aliases by the name of the library and the name of the variable, all instances of I/O variable aliases in your VI are automatically updated. Finally, delete the dummy shared variable that you created before the migration process.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Name	Type	VarType	Global	Global:En	Industrial	Industrial:	Industrial:	Network:/	Network:I	Network:I	Network:I	Network:I	Real-Time	Real-Time	Real-Time	Real-Time Features:
2	Thermoco	Double	Industrial			on	FALSE	2	read only	TRUE	CompactR	TRUE	FALSE				
3	Heater 1	Double	Industrial			on	TRUE	2	read/writ	TRUE	CompactR	TRUE	FALSE				
4	Thermoco	Double	Global	on	FALSE							FALSE	FALSE	on	FALSE	1	FALSE

1. Leave variable types alone. 2. Copy "dummy" shared variable settings to all IOVs. 3. Save variable list as .csv.

Figure 11.47. You can easily convert an I/O variable (IOV) alias library to shared variables by exporting the variables to a spreadsheet, modifying the parameters, and importing them into your new target.

The final step for implementing an FPGA scan engine is adding a real-time process to read data from the FPGA and constantly update the current value table. The FPGA I/O you are adding to the shared variables is deterministic, so you can use a Timed Loop for implementing this process.

To read data from the FPGA scan engine, create a Timed Loop task set to the desired scan rate in your top-level RT VI. This Timed Loop is the deterministic I/O loop, so you should set it to the highest priority. To match the control loop speed of your previous Scan Mode application, set the period of this loop to match the period previously set for Scan Mode. You also need to change the timing sources of any other task loops in your application that were previously synchronized to the Scan Mode to the 1 kHz clock and set them to the same rate as the I/O scan loop.

The I/O scan loop pushes new data to the FPGA and then pulls updated input values. The specific write and read VIs are also responsible for the scaling and calibration of analog and specialty digital I/O.

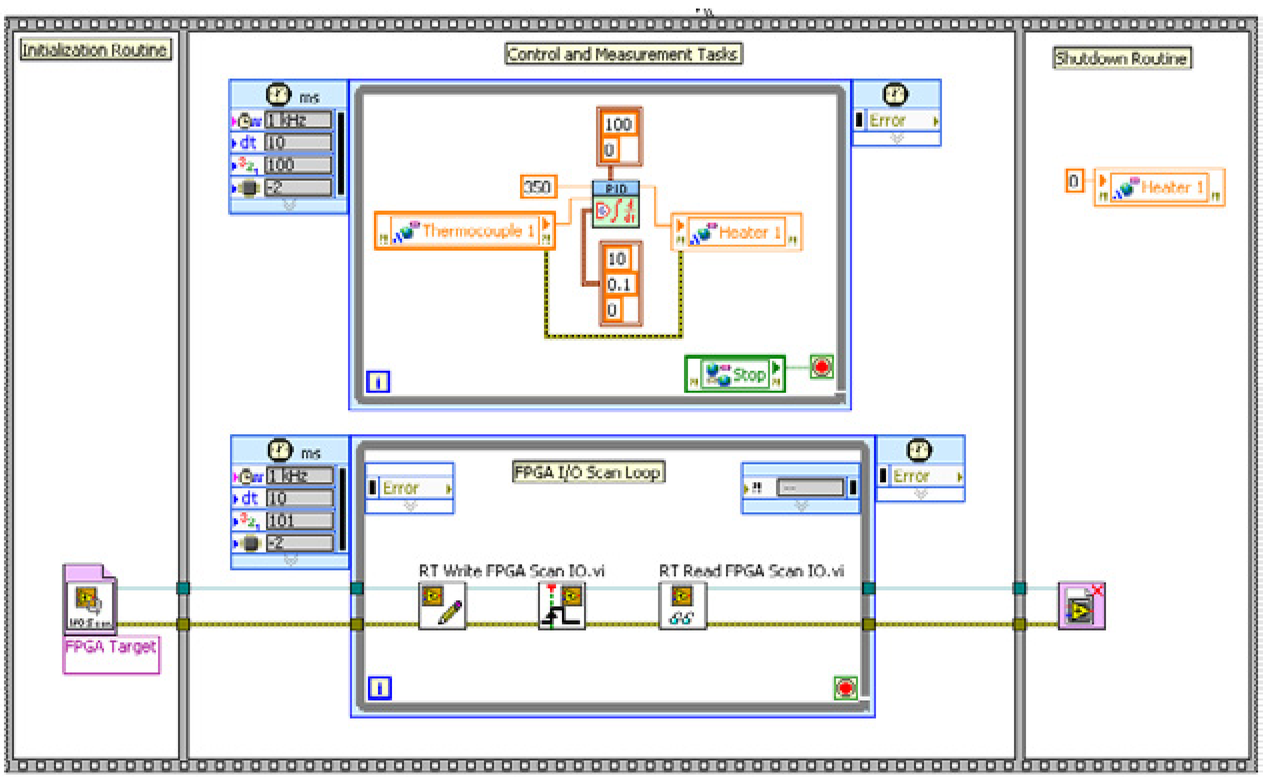


Figure 11.48. The FPGA I/O scan loop mimics the RIO Scan Interface feature by deterministically communicating the most recent input and output values to and from the FPGA I/O and inserting the data into a current value table.

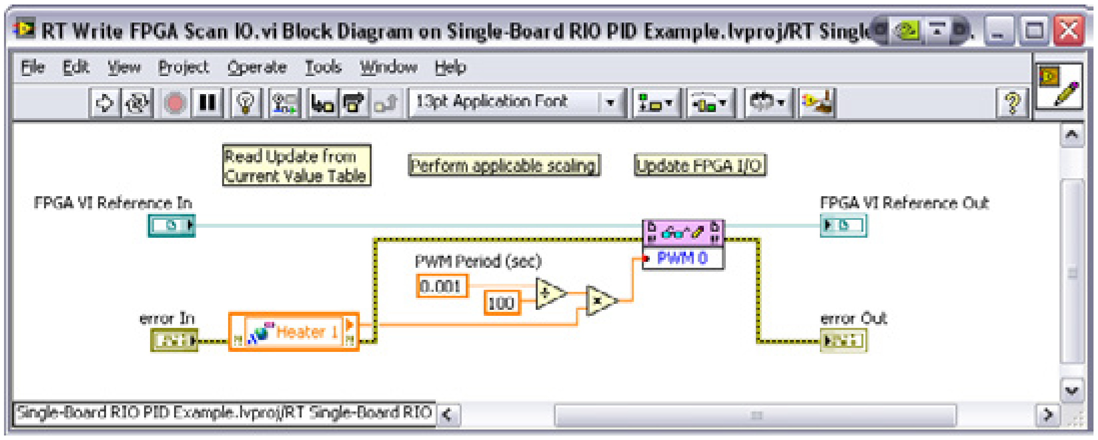

RT Write FPGA Scan IO.vi Block Diagram on Single-Board RIO PID Example.lvproj/RT Single
File Edit View Project Operate Tools Window Help
13pt Application Font
Read Update from Current Value Table
Perform applicable scaling
Update FPGA I/O
FPGA VI Reference In
FPGA VI Reference Out
PWM Period (sec)
0.001
100
PWM 0
error In
Heater 1
error Out
Single-Board RIO PID Example.lvproj/RT Single-Board RIO

Figure 11.49. The RT Write FPGA Scan IO VI pulls current data using a real-time FIFO single-process shared variable, scales values with appropriate conversion for the FPGA Scan VI, and pushes values to the FPGA VI.

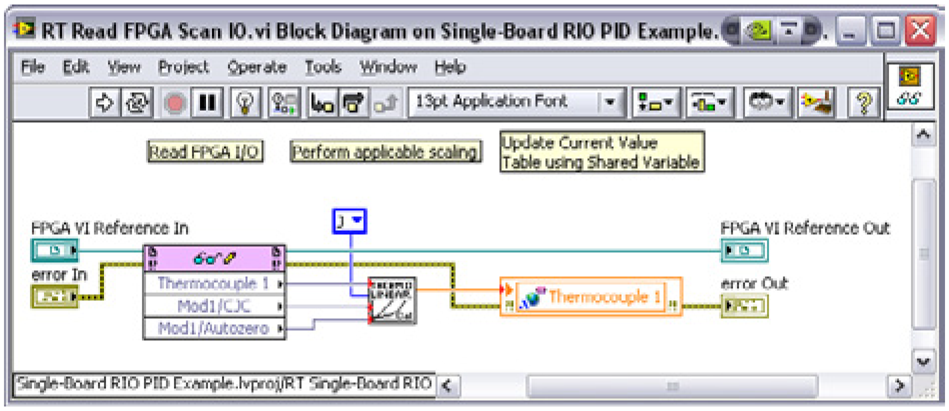

RT Read FPGA Scan IO.vi Block Diagram on Single-Board RIO PID Example.
File Edit View Project Operate Tools Window Help
13pt Application Font
Read FPGA I/O
Perform applicable scaling
Update Current Value Table using Shared Variable
FPGA VI Reference In
error In
Thermocouple 1
Mod1/CJC
Mod1/Autozero
Thermocouple 1
FPGA VI Reference Out
error Out
Single-Board RIO PID Example.lvproj/RT Single-Board RIO

Figure 11.50. The RT Read FPGA Scan IO VI pulls all updates from the FPGA Scan IO VI, performs applicable conversions and scaling, and publishes data to a current value table using a real-time FIFO single-process shared variable.

After building the host interface portion of a custom FPGA I/O scan to replace Scan Mode, you are ready to test and validate your ported application on the new target. Ensure the FPGA VI is compiled and the real-time and FPGA targets in the project are configured correctly with a valid IP address and RIO resource name. After the FPGA VI is compiled, connect to the real-time target and run the application.

Because the RIO architecture is common across NI Single-Board RIO, CompactRIO, and R Series FPGA I/O devices, LabVIEW code written on each of these targets is easily portable to the others. As demonstrated in this section, with proper planning, you can migrate applications between all targets with no code changes at all. When you use the specialized features of one platform, such as the RIO Scan Interface, the porting process is more involved, but only the I/O portions of the code require change for migration. In both situations, all of the LabVIEW processing and control algorithms are completely portable and reusable across RIO hardware platforms.

CHAPTER 12
Security on NI RIO Systems

Security Concerns on RIO Systems

Reconfigurable I/O (RIO) systems (NI Single-Board RIO, NI CompactRIO, and so on) are often used in critical applications. While the application space for the RIO platform is expansive, security concerns with RIO systems can be more narrowly defined.

One key security concern is functional correctness, wherein the I/O of the RIO device is proper. This is a security concern because if functional correctness is compromised, the hardware that the RIO device is connected to can get damaged or malfunction. For example, failed products on an assembly line may pass if the I/O of the RIO device has been compromised, or alternatively, motors and centrifuges controlled by the RIO device may get permanently damaged if they are ramped improperly.

In addition to functional correctness, a key security concern is sensitive data protection. RIO devices often compute, carry, or transmit sensitive data. It's important to ensure that this data is properly protected. Besides the data itself, valuable algorithms are often programmed onto RIO devices as well. Keeping these algorithms safe from theft is also crucial.

The three key security concerns, functional correctness, sensitive data protection, and algorithm protection are seldom considered independently on a RIO system. Often, a compromise in one of these areas leads to a compromise in another.

The Nature of Security on RIO Systems

Security is a complex challenge regardless of the system. It's not a measure that can be addressed once and then forgotten. It must be continually managed. Additionally, definitions of "secure" can vastly differ because they are extremely application specific. Security, fundamentally, comes at a trade-off; more secure systems require greater time and cost investments and sacrifice ease of use.

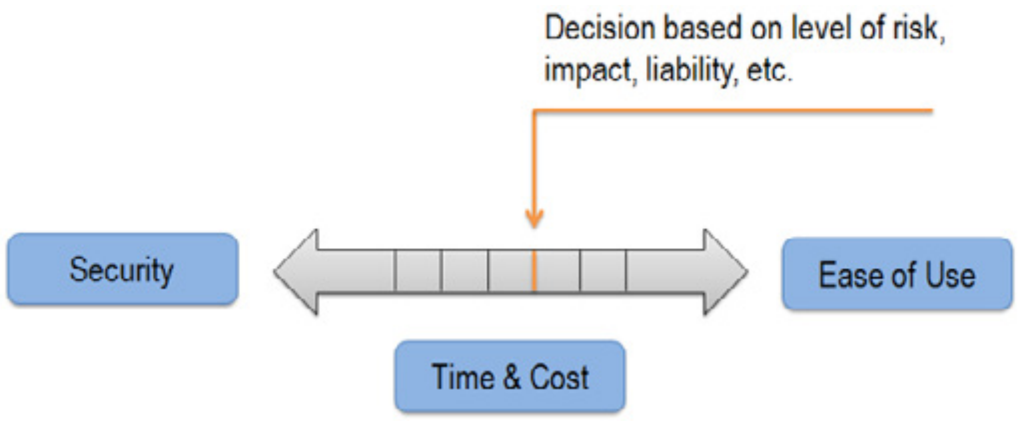


Figure 12.1. The Trade-Off Between Security and Time, Cost, and Ease of Use

Given this trade-off, you need to evaluate the proper investment in security for each application. The security required by an application depends on the liability and scope of the application. National Instruments provides best practices and open channels of communication to help you overcome the challenges that security presents.

Layers of Security

Security can be defined at several different levels in most systems. For a RIO system, security is defined at the following key levels; physical, network, operating system, and application. It's important to have some protection at each level, otherwise, a compromise in one layer can easily lead to a compromise in another. If you invest a lot of time and effort protecting one layer and fail to consider other layers, an intruder may find a way around the well-protected layer and manage to compromise your system with little effort.

First, you need to understand how the four RIO system layers are related. The layers farthest away from the application are the physical and network layers. Security threats to a RIO system can emerge from these two layers. The next affected layer is the operating system layer, which is closest to the application. Lastly, the application layer, which resides at the core, requires the most protection. Steps can be taken at each of the layers to ensure that the overall application is not compromised.

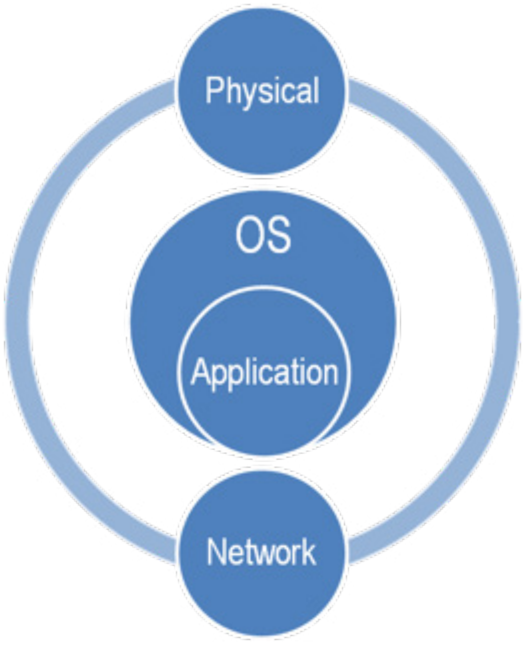


Figure 12.2. The Layers of Security Important to RIO Systems

Security Vectors on RIO Systems

A typical RIO system comprises a host PC and a RIO target, connected over a network, as shown in Figure 13.3. The host PC can be either the development or deployment machine—in some cases, these are the same. The network is often a private local area network (LAN), but it can also be as simple as a crossover connection. As mentioned, threats to the RIO system can originate at the physical or network level. The targets can be either the host PC or the RIO device. Thus you need to consider the threats that can affect both the host PC and the RIO device, and not focus solely on the RIO device itself. To avoid redundancy, the best practices documentation focuses on the threats to the RIO device that originate in the network between the host PC and the RIO and not with the threats to the host PC.

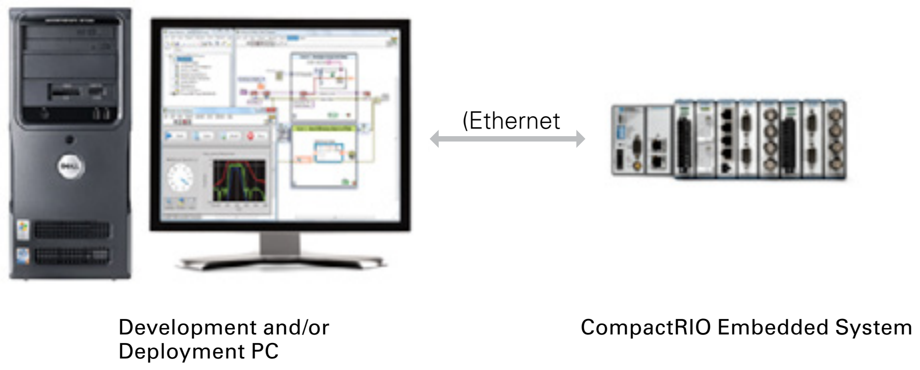


Figure 12.3. A Typical RIO Deployment System

Best Practices for Security on RIO Systems

The best practices for security on RIO systems are organized into three groups: recommended, optional, and extreme. The best practices presented online are not exhaustive; there are further measures you can take, but these are reserved for extremely advanced users. To learn about what you can do beyond the best practices presented in these online articles, contact National Instruments either through support or your local sales or account representative.

The recommended practices should be adopted by all users before deploying the RIO system. The optional practices take more time and effort to implement, but they provide the added security required by some applications. Lastly, the extreme section highlights a few measures that you can take to further secure a RIO system. Note that the extreme measures require a significant investment in time and effort and may not be appropriate for all applications.

Best Practices Resources

Best Practices for Security on RIO Systems: Part 1 Recommended

Best Practices for Security on RIO Systems: Part 2 Optional

Best Practices for Security on RIO Systems: Part 3 Extreme