CompactRIO Developers Guide

Recommended LabVIEW Architectures and Development Practices
for Machine Control Applications

Section Five
Deploying and Replicating Systems

This document provides an overview of recommended architectures and development practices when building machine control applications using NI CompactRIO controllers and NI touch panel computers.
The examples and architectures are built using NI LabVIEW versions 8.6 and 2009.

December 2009

NATIONAL INSTRUMENTS™

CONTENTS

CHAPTER 8
Deploying and Replicating Applications 195

Application Deployment 195

Deploying Applications to CompactRIO 195
Deploy a LabVIEW VI onto Volatile Memory 195
Deploy a LabVIEW VI onto Nonvolatile Memory 196

Deploying Applications to a Touch Panel 199
Configure the Connection to the Touch Panel 199
Deploy a LabVIEW VI onto Volatile or Nonvolatile Memory 199
Deploy an Executable Touch Panel Application to a Windows CE or XP Embedded Target 203

Deploying Applications that Use Network-Published Shared Variables 204
Network Shared Variable Background 204
Deploy Shared Variable Libraries to a Target That Hosts Variables 204
Undeploy a Network Shared Variable Library 206
Deploy Applications That Are Shared Variable Clients 206

Recommended Software Stacks for CompactRIO 208

System Replication 209

Real-Time Replication Utility VIs 209

Building a Replication Utility 211
NI-RIO System Replication Tools 211

IP Protection 212

Locking Algorithms or Code to Prevent Copying or Modification 212
Protect Deployed Code 212
Protect Individual VIs 212
Lock Code to Hardware to Prevent IP Replication 214
Acquire the MAC Address of Your CompactRIO System 215
CompactRIO Information Retrieval Tools 216

Porting to Other Platforms 217
LabVIEW Code Portability 217

NI Single-Board RIO 217
Port CompactRIO Applications to NI Single-Board RIO or R Series Devices 218
Example of Porting a CompactRIO Scan Mode-Based Application to NI Single-Board RIO 219

CHAPTER 8
Deploying and Replicating Applications

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.

Deployment onto Volatile Memory

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

Deployment onto Nonvolatile Memory

If you deploy the application onto the nonvolatile memory on a target, the application remains after you cycle the power on the target. It is also possible to set applications stored on nonvolatile memory to start up automatically when the target boots. This is useful when you have finished code development and validation and want to create a stand-alone embedded system.

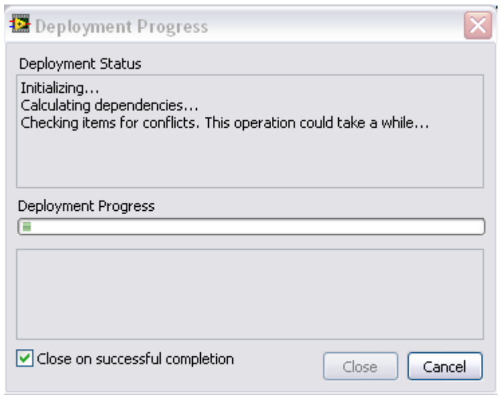
Deploying Applications to CompactRIO

Deploy a LabVIEW VI onto Volatile Memory

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

- Target the CompactRIO controller in LabVIEW
- Open a VI under the controller
- Press 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.



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

Figure 8.1. LabVIEW Deploying an Application onto the Nonvolatile Memory of the Controller

Deploy a LabVIEW VI onto Nonvolatile Memory

Once you have finished developing and debugging your application, you likely want to deploy your code onto the nonvolatile memory on the controller so that it persists through power cycles and configure the system so the application runs on startup. To deploy an application onto the nonvolatile memory, you first need to build the VI into an executable.

Building an Executable from a VI

The LabVIEW Project provides the ability to build an executable real-time application from a VI. To build an executable real-time application, you create a build specification under the real-time target in the LabVIEW Project Explorer. By right-clicking 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.

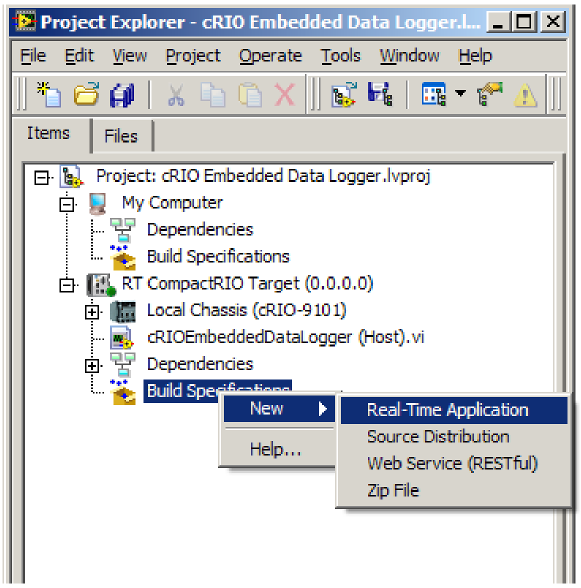
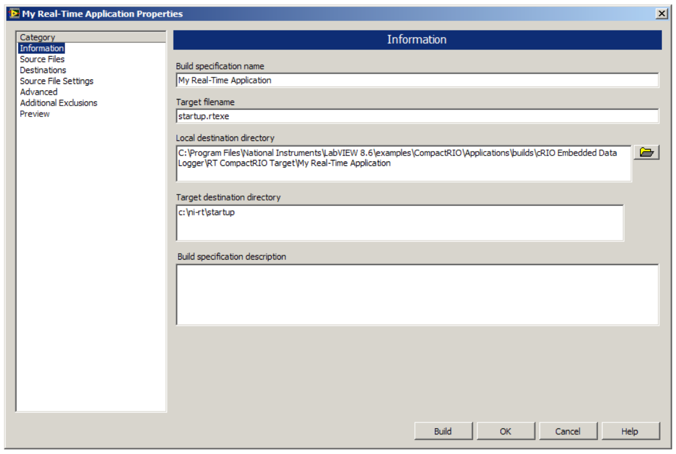

Project Explorer - cRIO Embedded Data Logger.l...
File Edit View Project Operate Tools Window Help
Items
Files
Project: cRIO Embedded Data Logger.lvproj
My Computer
Dependencies
Build Specifications
RT CompactRIO Target (0.0.0.0)
Local Chassis (cRIO-9101)
cRIOEmbeddedDataLogger (Host).vi
Dependencies
Build Specifications
New
Help...
Real-Time Application
Source Distribution
Web Service (RESTful)
Zip File

Figure 8.2. 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 Destinations, Source File Settings, Advanced, and Additional Exclusions categories are rarely used when building real-time applications.

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.



Build specification name
My Real-Time Application

Target filename
startup.rtexe

Local destination directory
C:\Program Files\National Instruments\LabVIEW 8.6\examples\CompactRIO\Applications\builds\cRIO Embedded Data Logger\RT CompactRIO Target\My Real-Time Application

Target destination directory
c:\ni-rt\startup

Build specification description

Figure 8.3. 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 VIs as Startup VIs or Always Included unless they are called dynamically in your application.

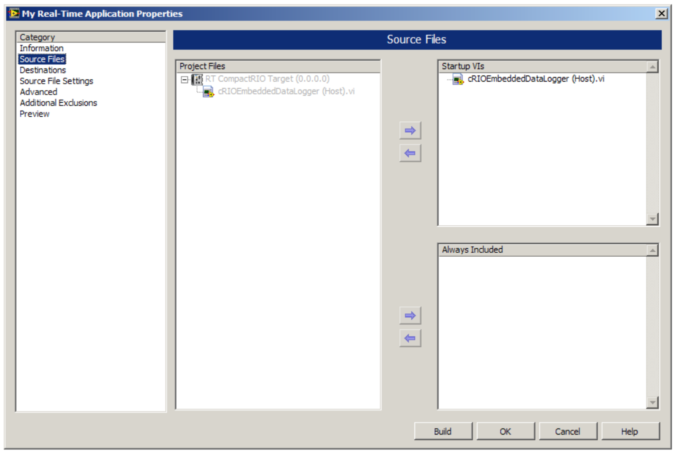


Figure 8.4. Source Files Category in the Real-Time Application Properties (In this example, the cRIOEmbeddedDataLogger (Host).vi was selected to be a Startup VI.)

After all 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.

Getting 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.

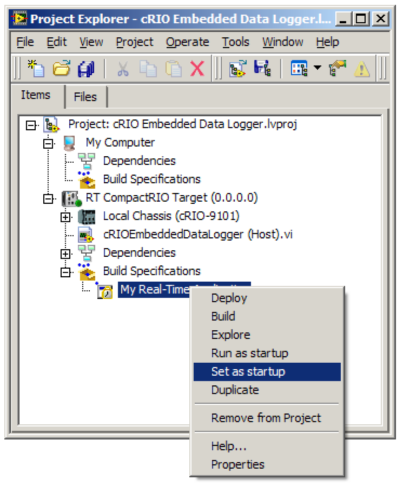


Figure 8.5. Configuring a Build Specification to Run When an Application Boots

Deploy Executable Real-Time Application 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 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. Behind the scenes, LabVIEW copies the executable files onto the nonvolatile memory on the controller 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.

At some point, you may want to remove an executable you stored on your real-time target. The easiest way to do this is to use FTP to access the real-time target and delete the executable file that was deployed to the target. If you used the default settings, the file is located in the NI-RT\Startup folder with the name supplied in the target filename box from the Information category and the extension .rtexe.

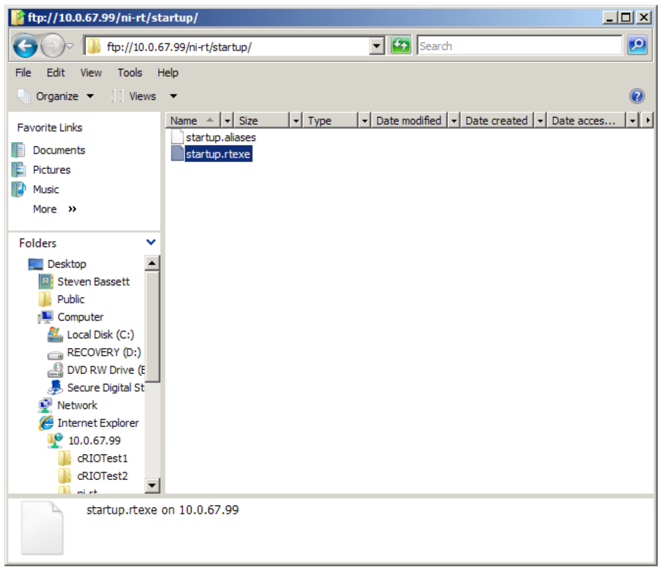
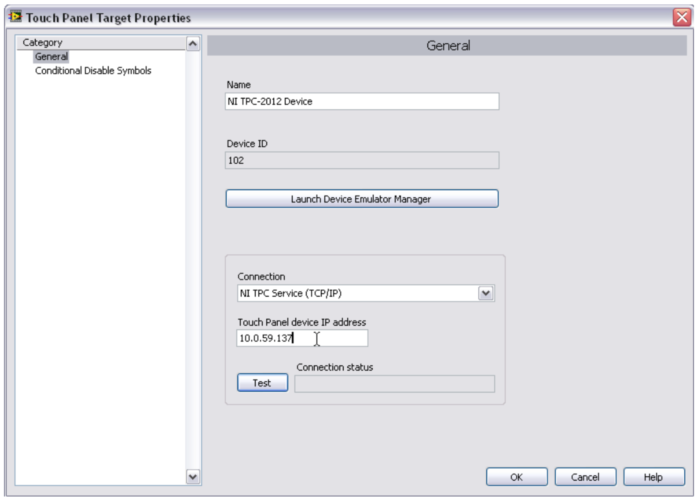


Figure 8.6. Deleting the startup.rtexe from a CompactRIO Controller

Deploying Applications to a Touch Panel

Configure the Connection to the Touch Panel

Although it is possible to manually copy built applications to a touch panel device, it is recommended that you use Ethernet and allow the LabVIEW Project to automatically download the application. National Instruments touch panels all ship 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 connection as NI TPC Service 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

Name: NI TPC-2012 Device

Device ID: 102

Connection: NI TPC Service (TCP/IP)

Touch Panel device IP address: 10.0.59.137

Connection status:

Figure 8.7. 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 command prompt on the TPC and typing ipconfig. To get to the command prompt go to the Start menu and select Run… In the popup window enter cmd.

Deploy a LabVIEW VI onto Volatile or Nonvolatile Memory

The steps to deploy an application onto a Windows XP Embedded touch panel and onto a Windows CE touch panel are nearly identical. The only difference is on an XP Embedded touch panel , you can deploy an application onto only the nonvolatile memory, and, on a Windows CE touch panel, you can deploy onto 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 build an executable Touch Panel Application you create a build specification under the touch panel target in the 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.

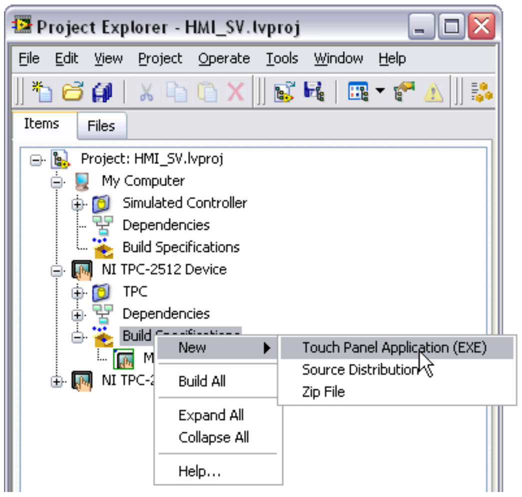
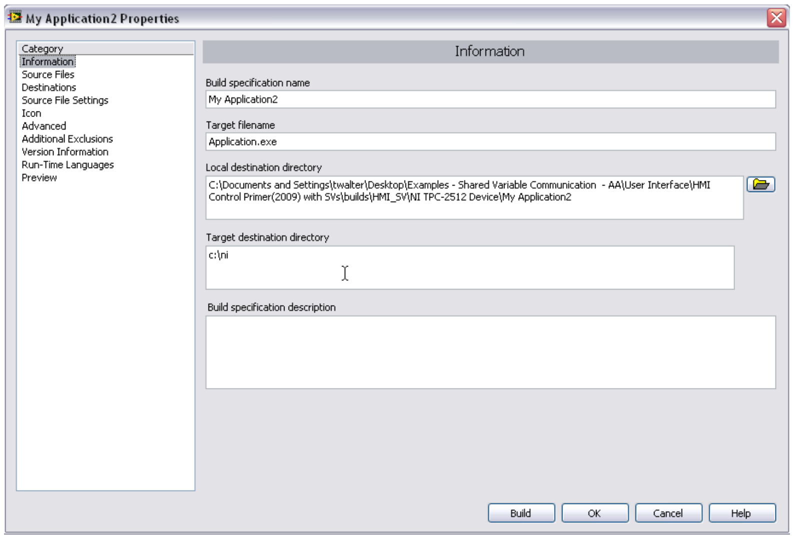


Figure 8.8. Create a touch panel application using the LabVIEW Project.

After selecting the **Touch Panel Application**, you are presented with a dialog box. The two main 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.



My Application2 Properties

Category: Information, Source Files, Destinations, Source File Settings, Icon, Advanced, Additional Exclusions, Version Information, Run-Time Languages, Preview

Information

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 8.9. The Information Category in the Touch Panel 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 VIs as Startup VIs or Always Included unless they are called dynamically in your application.

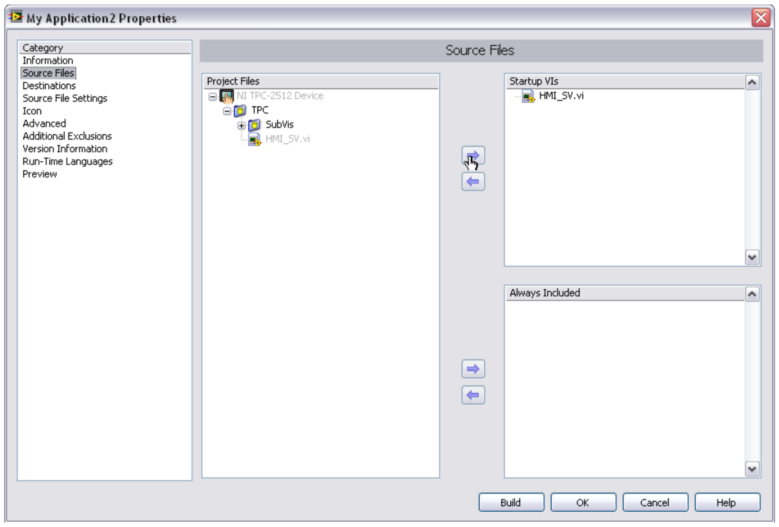


Figure 8.10. Source Files Category in the Touch Panel Application Properties (In this example, the HMI_SV.vi was selected to be a Startup VI).

After all 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.

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.

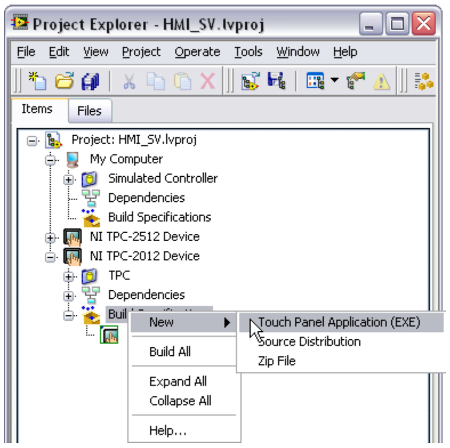
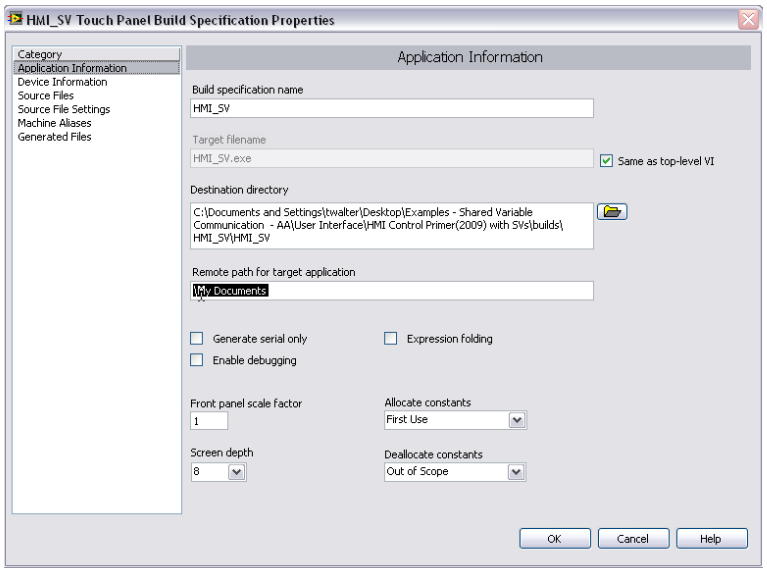


Figure 8.11. Creating a Touch Panel Application Using the LabVIEW Project

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



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

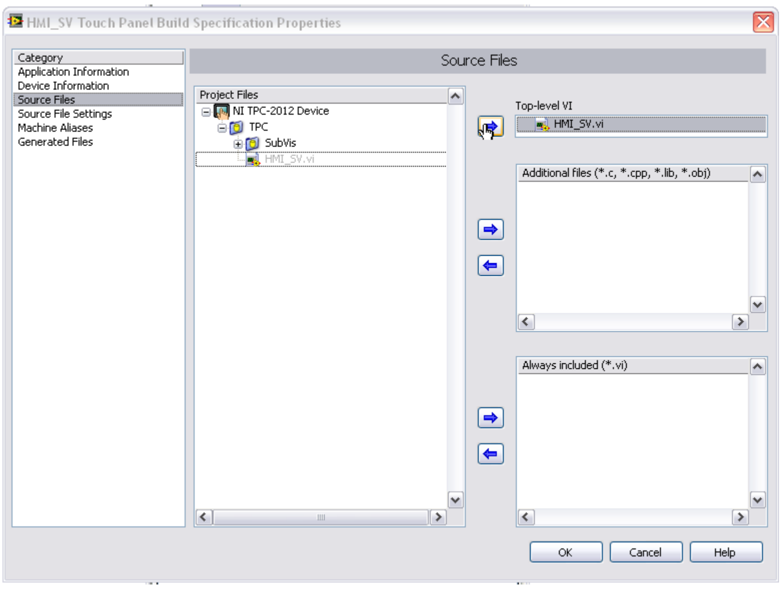
Remote path for target application: \My Documents

☐ 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 8.12. The Information Category in the Touch Panel Application Properties

The Source Files category is used to set the startup VI and include 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.



Source Files

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

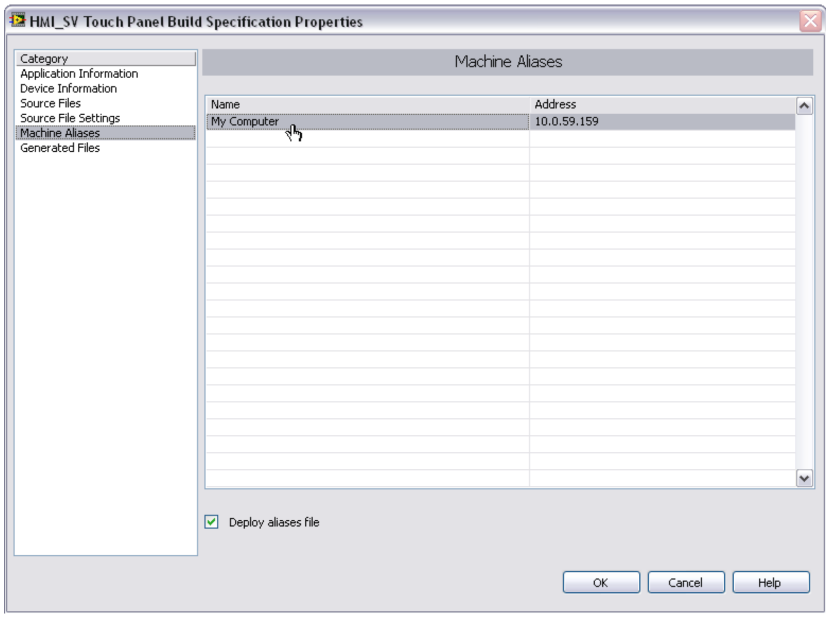
Top-level VI: HMI_SV.vi

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

Always included (*.vi)

Figure 8.13. 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" check box. The alias list should include your network-published shared variable servers and their IP addresses (normally CompactRIO or Windows PCs). More details on alias files and deploying applications using network-published shared variables are covered in the deployment section of this document.



Name	Address
My Computer	10.0.59.159

☑ Deploy aliases file

Figure 8.14. The Machine Aliases Category in the Touch Panel Application Properties
(Be sure to check the deploy aliases file check box if using network-published shared variables.)

After all 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.

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 onto 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 startup on boot. This utility modifies the startup.ini file for you.

Deploying Applications that Use Network-Published Shared Variables

Network Shared Variable Background

The term **network shared variable** refers to a software item that exists on the network and can communicate between programs, applications, remote computers, and hardware.

There are three pieces that make the network variable work in LabVIEW.

Network Variable Nodes

A network variable node is the block diagram representation for network reads and writes. Each variable node references a software item on the network (the actual network variable) hosted by the Shared Variable Engine. Figure 8.15 shows a network variable node, its actual network path, and its respective item in the project tree.

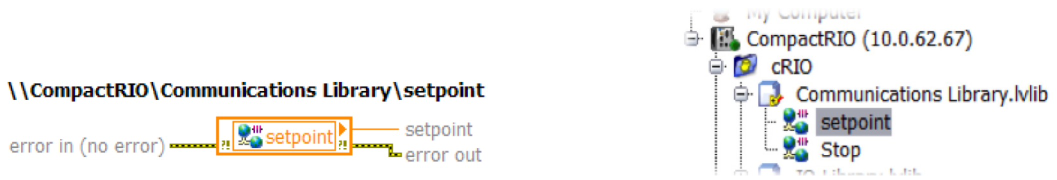


Figure 8.15. Network Variable Node and Its Network Path

Shared Variable Engine

The Shared Variable Engine is a software component that publishes data over Ethernet. The engine must run on real-time targets or Windows PCs where network shared variables are hosted. On Windows, the Shared Variable Engine is a service launched at system startup. On a real-time target, it is a driver that loads when the system boots.

When the shared variable engine starts it reads data stored on the nonvolatile memory to determine what variables it should publish on the network.

Publish-Subscribe Protocol (PSP)

The Shared Variable Engine uses the NI Publish-Subscribe Protocol (NI-PSP) to communicate data. The NI-PSP is a networking protocol built using TCP where each shared variable client subscribes to data hosted by a Shared Variable Engine.

Deploy Shared Variable Libraries to a Target That Hosts Variables

A CompactRIO system starts the Shared Variable Engine when it boots, and the engine accesses the nonvolatile memory to determine what if any libraries it needs to deploy. Shared variable libraries are automatically deployed when you run a VI that accesses any of the variables and when you deploy an application that accesses any of the variables. However, it is possible that no libraries have ever been deployed to the system. If this is the case, the engine does not make any variables available on the network.

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 variable engine to create new data items on the network.

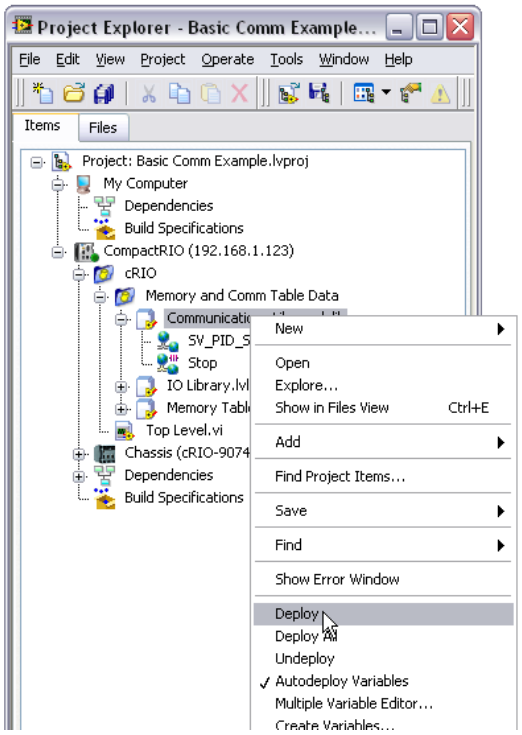


Figure 8.16. Deploy libraries to real-time targets by selecting Deploy from the right-click menu.

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 bring up the programming palette and go to Programming » Application Control and place the Invoke Node on the block diagram
    - Using the hand tool, click on Method and select Library » Deploy Library

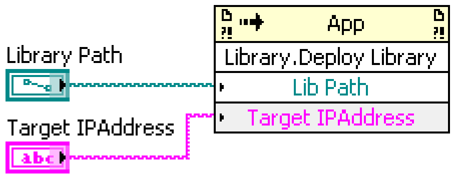


Figure 8.17. You can programmatically deploy libraries to real-time targets using the application invoke node on a PC.

- 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 a library is deployed to a Shared Variable Engine, those settings persist until you manually undeploy them. To undeploy a library:

1. Launch the NI Distributed System Manager (From **LabVIEW»Tools** or from the Start Menu)
2. Add the real-time system to "My Systems" (Actions»Add System to My Systems)
3. 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.

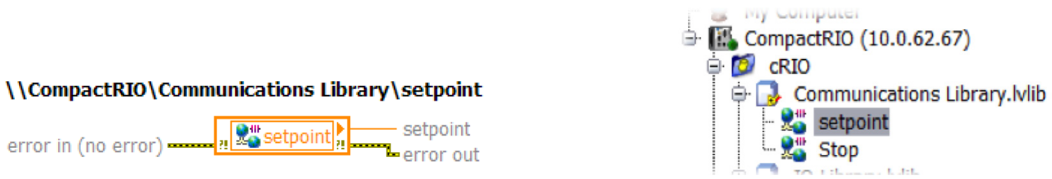
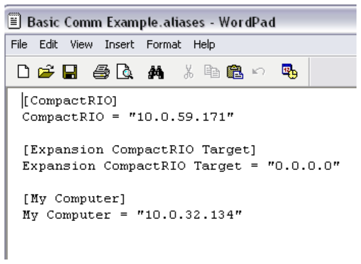


Figure 8.18. Network Variable Node and Its Network Path

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 only need to edit 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

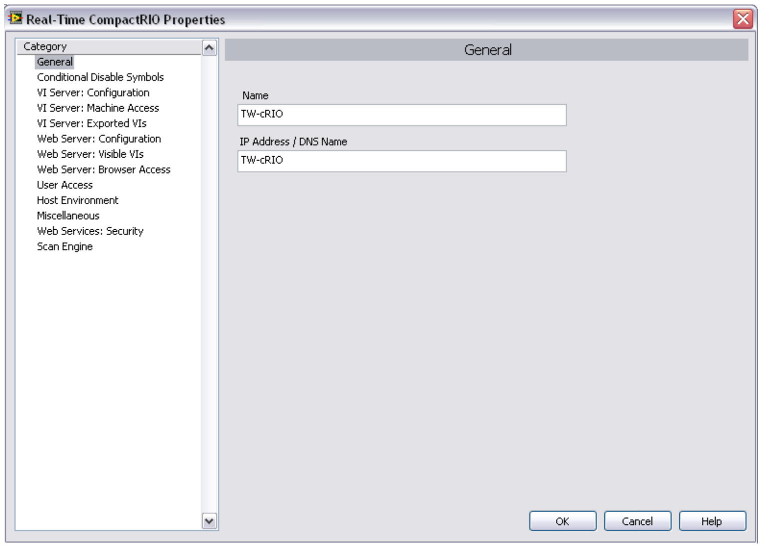
```
[CompactRIO]
CompactRIO = "10.0.59.171"

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

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

Figure 8.19. The Alias file is a human-readable file that lists the target name and IP address.

If you are deploying systems that have dynamic IP addresses using DHCP, you can use the 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.



General

Name
TW-cRIO

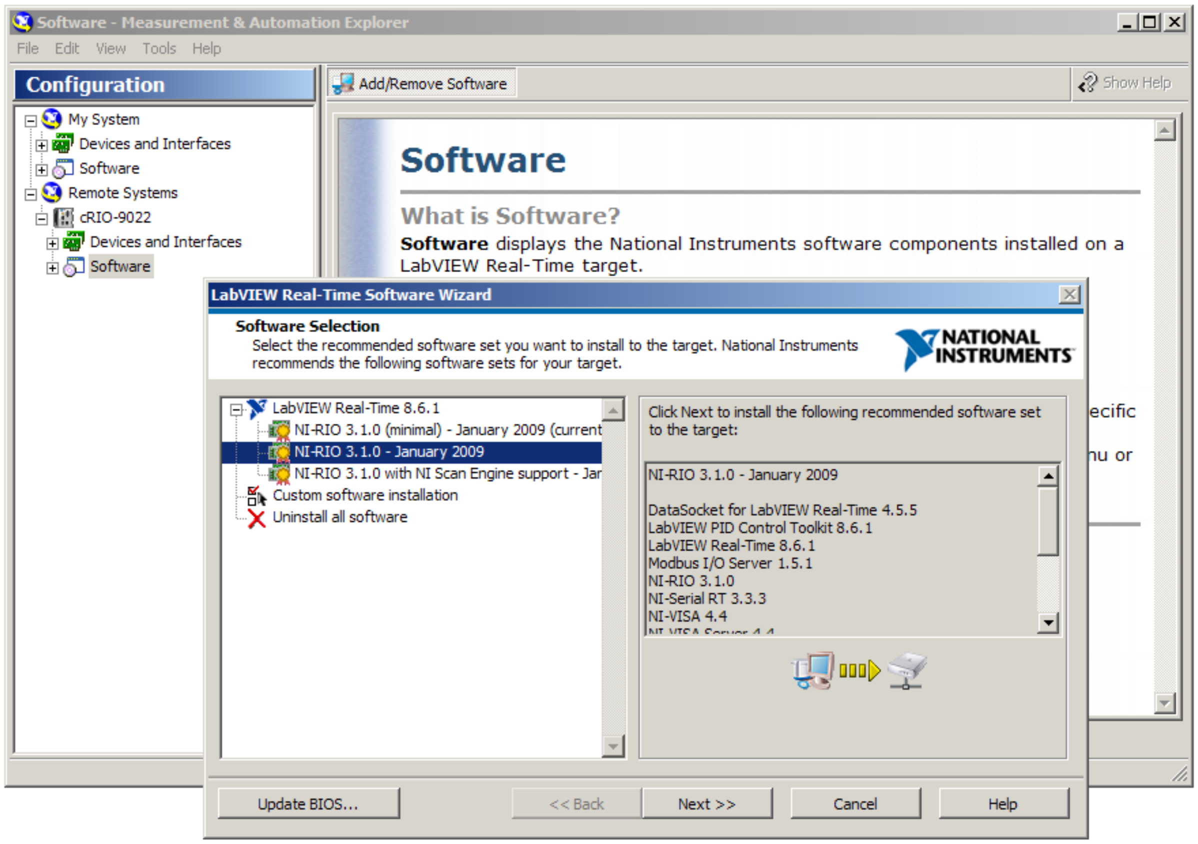
IP Address / DNS Name
TW-cRIO

Figure 8.20. 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 don't exist) with a name indicating its purpose in the application. Then as part of the installer, you can run an executable which 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

National Instruments also provides several sets of commonly used driver sets called Recommended software sets. Recommended software sets can be installed onto CompactRIO controllers from Measurement and Automation Explorer.



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.

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 8.21. 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.

System Replication

Example

LabVIEW Example Code
is provided for this section

After you have deployed a LabVIEW Real-Time application to a CompactRIO controller, you may want to deploy that image to other identical real-time targets. You can use the LabVIEW Project and the LabVIEW Application Builder to redeploy an already built application with the procedure described above. This method of replication becomes cumbersome when attempting to replicate and deploy more than a few systems.

To help speed deployments, National Instruments provides a set of system replication VIs for the replication of LabVIEW Real-Time targets. You can use these tools to copy the contents of a LabVIEW Real-Time controller hard drive and then replicate the information onto multiple controllers. You also can use these tools to programmatically identify systems on the network and configure network settings. This eliminates the use of Measurement & Automation Explorer (MAX) and an FTP client.

Real-Time Replication Utility VIs

Starting in LabVIEW 2009, eight VIs are installed with LabVIEW Real-Time for system replication.

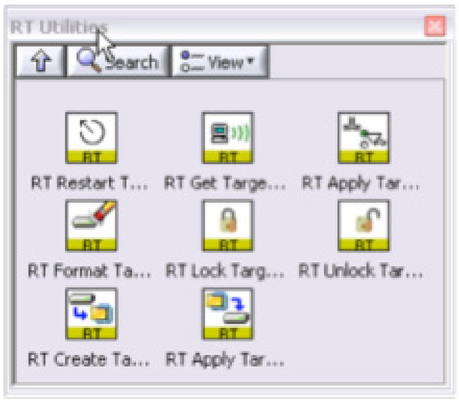


Figure 8.22. With RT Utility VIs, you can achieve programmatic controller configuration and imaging.

With the RT Get Target Information VI, you can search the network for real-time devices. It can find devices by IP address and MAC address, or it can find all NI real-time devices located on the same subnet.

Once the RT Get Target Information VI locates a programmed target on the network, the RT Create Target Disk Image connects to the target and transfers all the contents via FTP to your Windows computer. These contents are stored on your local machine in a zip file. The process of copying all the information from the controller takes a few minutes.

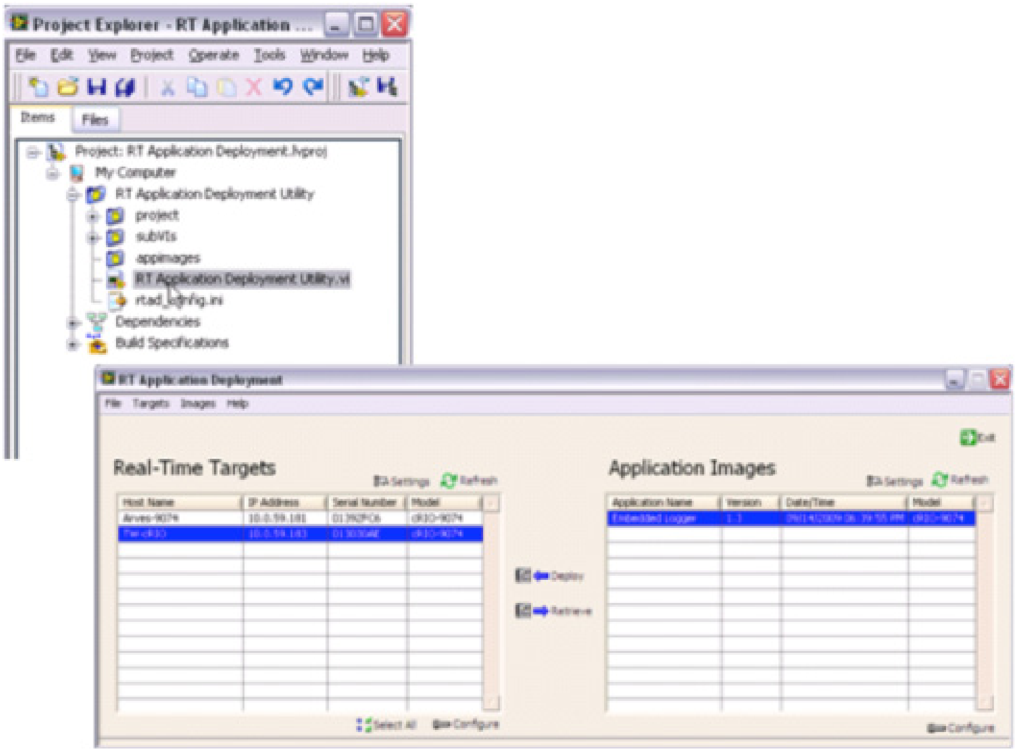
You can configure new controllers using the RT Apply Network Settings and the RT Apply Target Disk Image VIs. The first VI sets the network settings on the target and the second takes a previously stored disk image and downloads the contents to the new real-time controller. The process of copying a disk image to the controller takes a few minutes.

VIs also can help you lock and unlock the FTP server on the real-time controller using a password.

Building a Replication Utility

With these VIs, you can build your own custom replication utility in LabVIEW running on a Windows machine.

A prebuilt replication example using these VIs is included in this guide. When you run the application from the desktop, it provides a graphical user interface for replicating and managing controllers and images.



Host Name	IP Address	Serial Number	Model
Arves-9074	10.0.59.181	01392FC6	cRIO-9074
[illegible]	10.0.59.183	[illegible]	cRIO-9074

Application Name	Version	Date/Time	Model
Embedded Logger	1.1	[illegible]	cRIO-9074

Figure 8.23. You can create sophisticated replication programs such as this example replication application.

This example application automatically scans the network for real-time targets and can scan the target directory for stored application images. You can retrieve target disk images and store them on the Windows machine and deploy application images to targets. The process of deploying or retrieving disk images takes a few minutes. You can find more information on using this example in the readme.doc file included in the LabVIEW Project.

NI-RIO System Replication Tools

NI-RIO System Replication Tools offer additional support to Real-Time Replication Tools by providing a LabVIEW API with functionality to programmatically download the FPGA bitfile to flash memory on the backplane. This is useful if you have a deployed application where the FPGA code needs to start running immediately and cannot be loaded in the standard way from the real-time code. With these tools, you can erase and download an FPGA bitfile to flash memory and set how a VI is loaded from flash memory. NI-RIO System Replication Tools require NI-RIO to be installed on the host PC using the VIs.

Set RIO Device Settings.vi

Use Set RIO Device Settings to configure when the bitfile loads from flash memory.

Download Bitfile.vi

This VI downloads a specific bitfile to one or more FPGA targets or erases the existing bitfile. The input is an IP address of a host machine, the FPGA target resource (RIO0), the path to the bitfile, and the operation to perform (download or erase the bitfile).

IP Protection

Intellectual property (IP) in this context refers to any unique software or application algorithm(s) that you 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 provides companies with a way to differentiate from competition. Therefore, protecting this software IP is very important. LabVIEW development tools and CompactRIO provide you the ability to protect and lock your IP. In general there are two levels of IP protection you can implement:

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

If you have created algorithms for specific functionality, such as performing advanced control functions, implementing custom filtering, and so on, you may want to be able to distribute the algorithm as a subVI but prevent someone from viewing or modifying that actual algorithm. This may be for IP protection or it may be 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

If you want to ensure that a competitor can't replicate your system by running your code on another CompactRIO system or may 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 start-up application on a CompactRIO controller is by default locked and cannot be opened. Unlike other off-the-shelf controllers or some PLCs where the raw source code is stored on the controller and only protected 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 bit file and cannot be "decompiled" back to LabVIEW code. To aid in future debugging and maintenance it is possible to store the LabVIEW project on the controller or to 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.

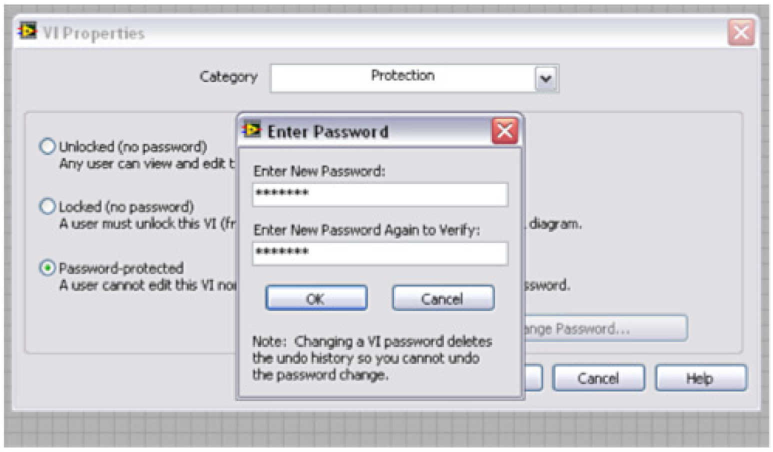
Protect Individual VIs

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 provides a few mechanisms to provide usable subVIs while still 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 subVI prohibits others from editing the VI or viewing its block diagram without the password. However, if the password is lost, there is no way you can 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 t

( ) Locked (no password)
A user must unlock this VI (fr

(●) Password-protected
A user cannot edit this VI no

diagram.

ssword.

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 8.24. Password Protecting LabVIEW Code

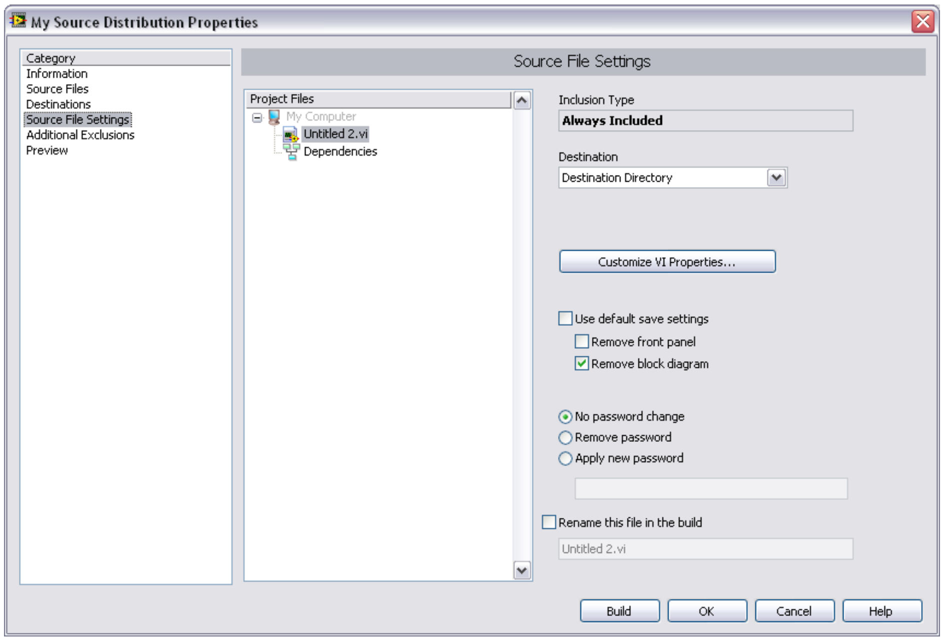
The LabVIEW password mechanism is quite difficult to defeat, but no password algorithm is 100% secure from attack. If you need total assurance that someone cannot gain access to your source code, you should consider removing the block diagrams.

Method 2. Removing the Block Diagram

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. Don't forget to make a backup of your files if you use this technique, as the block diagrams 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.

1. 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.
2. 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.
3. Build the source distribution to create a copy of the VI without its block diagram.


My Source Distribution Properties

Category
- Information
- Source Files
- Destinations
- Source File Settings
- Additional Exclusions
- Preview

Source File Settings

Project Files
- My Computer
  - Untitled 2.vi
  - Dependencies

Inclusion Type: Always Included

Destination: Destination Directory

Customize VI Properties...

- [ ] 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 8.25. Removing the Block Diagram from LabVIEW VIs

CAUTION: 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 easily 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 of 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 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

The following steps can be used 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 procedures below for more information 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 piece of CompactRIO hardware.

License Key

When deploying many systems hard coding the hardware identification may not be ideal as it requires a manual change to the source code and recompile for each system deployed. This problem can be addressed by using a license key file which is stored separate from the application code on the CompactRIO controller. The license key file can be easily updated for each system without needing to change the application. In addition to reading the MAC or serial number the VI can open the license file and verify that the license is valid. For security, the license file should be specific for each deployed system and your code should perform a mathematical operation between the license key and hardware specific features such as the MAC address. Because you can find the MAC and serial numbers programmatically on Windows and real-time OSs, you can develop a LabVIEW application for system deployment that automatically queries the CompactRIO system and generates and deploys the appropriate license key file.

Acquire the MAC Address of Your CompactRIO System

LabVIEW Example Code
is provided for this section

You can acquire the MAC address of real-time controllers manually from MAX on the network settings tab of the controller or from the Windows command line. You can also find the MAC address programmatically.

Acquiring the MAC Address from Windows

To programmatically find the MAC address of a real-time system from Windows, perform the following steps:

1. Run the RT Ping Controllers VI found in the **Real-Time»Real-Time Utilities** palette. This VI returns the network information of all real-time controllers on the subnet.

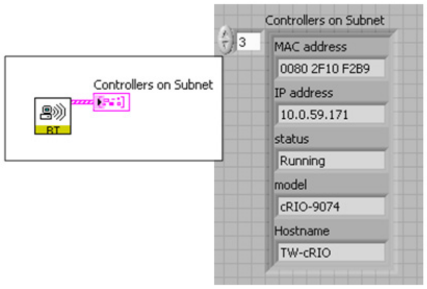


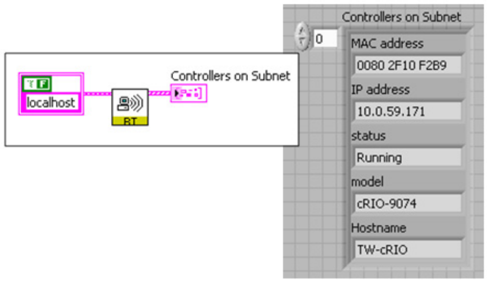
Figure 8.26. Obtain MAC address of networked real-time targets from Windows.

2. Search in the data that the RT Ping Controllers VI returned to find the appropriate real-time controller by using either the IP Address or Hostname.
3. This cluster returns the controllers' MAC addresses. The MAC addresses returned by the RT Ping Controllers VI are an array of strings in hexadecimal format.

Acquiring the MAC Address from the Real-Time Target

To find the MAC address programmatically from a real-time controller, perform the following steps:

1. Run the RT Ping Controllers VI found in the **Real-Time»Real-Time Utilities** palette while targeted to a real-time controller. Wire into the system location input a constant cluster with False and local host entries. This returns the information for the real-time controller where the code is executing.



Controllers on Subnet
0
MAC address
0080 2F10 F2B9
IP address
10.0.59.171
status
Running
model
cRIO-9074
Hostname
TW-cRIO

Figure 8.27. Obtain the MAC address on a real-time target.

2. Search in the data that the RT Ping Controllers VI returned for the controllers' MAC addresses. The MAC addresses returned by the RT Ping Controllers VI are an array of strings returned in a hexadecimal format.

CompactRIO Information Retrieval Tools

You can also get other information about CompactRIO system such as serial numbers using the CompactRIO Information Retrieval Tools. These free VIs can be run under Window or on the real-time controller 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.

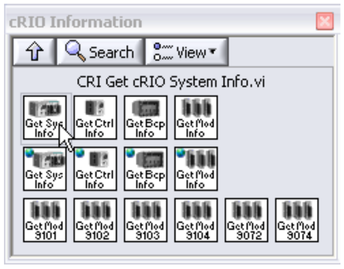


Figure 8.28. The CompactRIO Information Retrieval Tools return information such as serial numbers.

If you are using the LabVIEW Real-Time System Replication Tools, these also programmatically retrieve the real-time target's serial number.

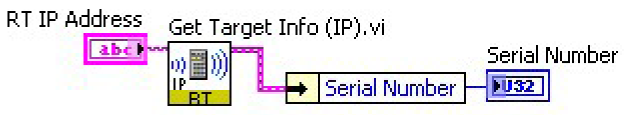


Figure 8.29. Obtain Serial Number Using Real-Time System Replication Tools

Porting to Other Platforms

This document has focused on architectures for building embedded control systems using CompactRIO systems. The same basic techniques and structures also work on other National Instruments 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.

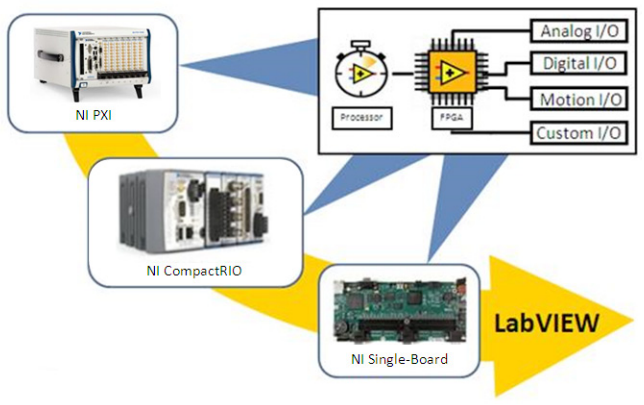


Figure 8.30. Using LabVIEW, you can use the same architecture for applications ranging from CompactRIO to high-performance PXI to board-level NI Single-Board RIO.

LabVIEW Code Portability

LabVIEW is a cross-platform programming language capable of compiling for multiple processor architectures and operating systems. In most cases, algorithms written in LabVIEW are portable between all LabVIEW targets. In fact you can even take LabVIEW code and compile it for any arbitrary 32 bit processor allowing you 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

The NI Single-Board RIO is a board-only version of CompactRIO designed for applications where a bare board form factor is required. While it is physically a different design, it uses the processor and FPGA and accepts up to three C Series modules. NI Single-Board RIO differs from CompactRIO because it includes I/O built directly onto the board. NI Single-Board RIO features 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.

LabVIEW FPGA Programming

NI Single-Board RIO does not currently support scan mode. Instead of using the 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 a memory table. This section examines an effective FPGA architecture for single-point I/O communication similar to scan mode later in this section and shows how to covert an application using scan mode.

Built-In I/O and I/O Modules

Depending on the I/O requirements of your application, 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 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)

NI Single-Board RIO 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 Xilinx 2M system gate Spartan-3 FPGA. CompactRIO targets offer versions using both the Spartan-3 FPGAs and larger, faster Virtex-5 FPGAs. 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-3 FPGA.

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

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

1. Build an NI Single-Board RIO or R Series project with equivalent I/O channels
2. If using CompactRIO Scan Mode, build a LabVIEW FPGA-based scan API
    a. Build LabVIEW FPGA I/O scan (analog in, analog out, digital I/O, specialty digital I/O)
    b. Convert I/O variable aliases to single-process shared variables with real-time FIFO enabled
    c. Build a real-time I/O scan with scaling and shared variable-based current value table
3. Compile LabVIEW FPGA VI for new target
4. Test and validate updated real-time and FPGA code

The first step in porting an application from CompactRIO to NI Single-Board RIO or 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 data acquisition (DAQ) device.

Step 2 is necessary only if the application being ported was originally written using scan mode. If you need to replace the scan mode portion of an application with an I/O method supported on NI Single-Board RIO and R Series, an example is included below to guide you through the process.

If the application you are migrating to NI Single-Board RIO did not use scan mode, 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 upon the RIO architecture and reusable modular C Series I/O modules, porting applications between these two targets is very simple.

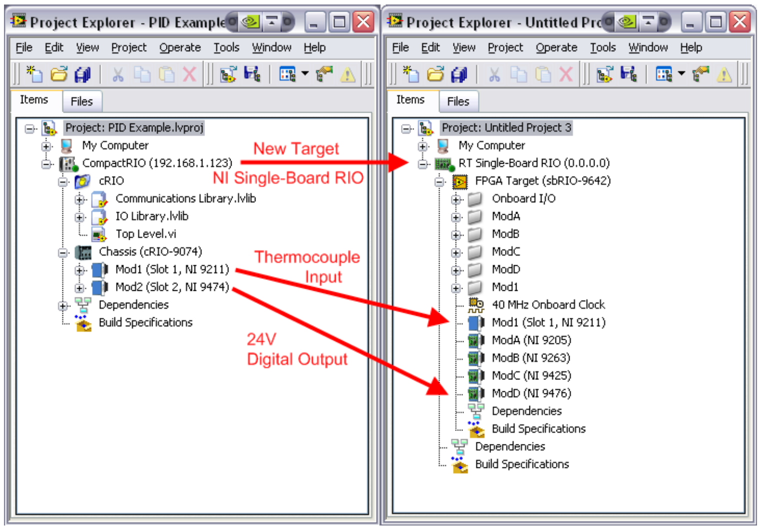


Figure 8.31. The first step in porting an application from CompactRIO to an alternate target is finding replacement I/O on the future target.

Example of Porting a CompactRIO Scan Mode-Based Application to NI Single-Board RIO

LabVIEW Example Code
is provided for this section

If you used scan mode in your original application, you need to create a simplified FPGA version of the scan mode because NI Single-Board RIO and R Series DAQ devices do not support scan mode. Building a scan engine in FPGA is very similar to the method for inserting single-point data from FPGA into the real-time scan discussed in the "Programming with LabVIEW FPGA" section. There are three steps to replace scan mode with a similar FPGA-based scan engine and current value table:

1. Build a LabVIEW FPGA I/O scan engine
2. Replace IOVs with single-process shared variables
3. Insert FPGA data into the shared variable based 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 encoder.

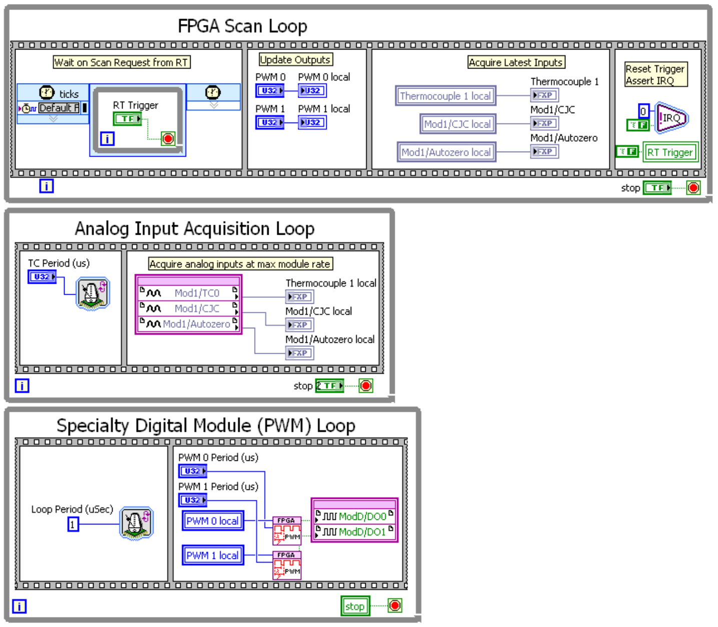
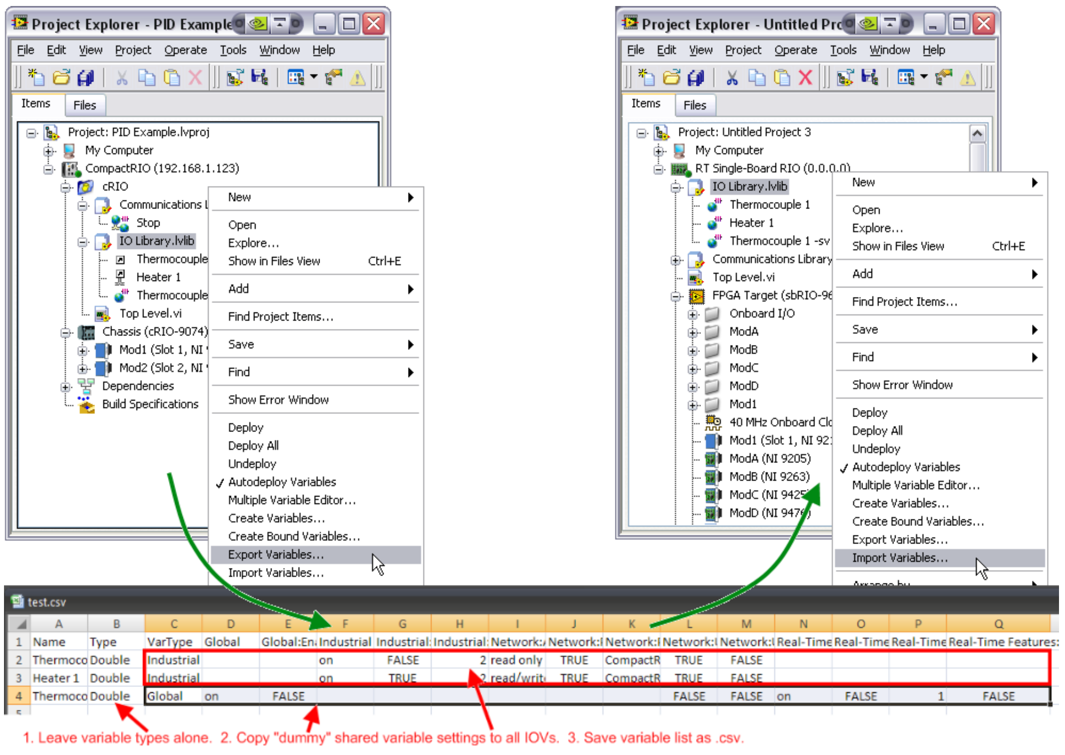


Figure 8.32. Develop a simple FPGA application to act as an FPGA scan engine.

After you have implemented a simple scan engine in the FPGA, it is time to port the real-time portion of the application to communicate with the custom FPGA scan rather than the current value table built into scan mode. 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, a single-process shared variables are not updated by a driver. You can change the type by going to the properties page for each IOV Alias and changing to single-process.

Tip: If you have numerous variables to convert, by exporting to a text editor and changing the properties you can easily convert a library of IOV aliases to shared variables. To make sure you get the properties correct it is easiest if you first create one “dummy” single-process shared variable with single element real-time FIFO enabled in the library then export the library to a spreadsheet editor. While in the spreadsheet editor, delete the columns exclusive to IOVs and copy the data exclusive to the share variables to the IOV rows. Then import the modified library into your new project. The IOV Aliases are imported as single-process shared variables. Because LabVIEW references shared variables and IOV Aliases by the name of the library and the name of the variable, all instances of IOV Aliases in your VI are automatically updated. Finally, delete the dummy shared variable that was 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 8.33. You can easily convert an IOV Alias Library to shared variables by exporting the variables to a spreadsheet, modifying the parameters, and importing into your new target.

The final step for implementing an FPGA based scan engine and shared variable current value table is building the real-time task to read data from the FPGA and constantly update the current value table. The FPGA I/O you are adding to the current value table is deterministic, so you again use the method described in the "Programming with LabVIEW FPGA" section, except for now you create the real-time portion of that code.

To read data from the FPGA based 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 it should be set 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. Any other task loops in your application that were previously synchronized to the scan also need to change their timing source to the 1 kHz clock and set to the same rate as the I/O 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 scaling and calibration of analog and specialty digital I/O.

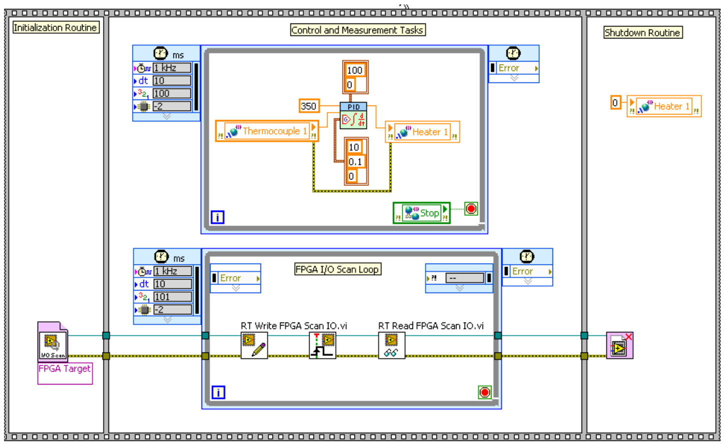


Figure 8.34. The FPGA I/O Scan Loop mimics the CompactRIO Scan Mode feature by deterministically communicating the most recent input and output values to and from the FPGA I/O and inserting the data into a memory table.

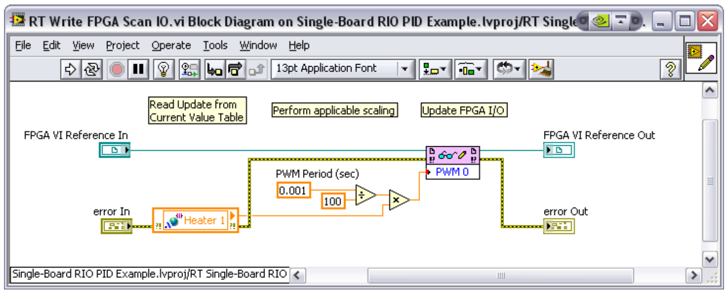

RT Write FPGA Scan IO.vi Block Diagram on Single-Board RIO PID Example.lvproj/RT Single
File Edit View Project Operate Tools Window Help
13pt Application Font
Read Update from Current Value Table
Perform applicable scaling
Update FPGA I/O
FPGA VI Reference In
FPGA VI Reference Out
PWM Period (sec)
0.001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

Figure 8.35. The RT Write FPGA Scan IO VI pulls data from the memory table 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.

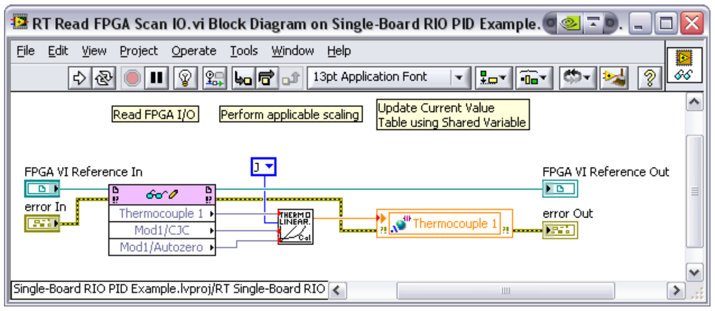

RT Read FPGA Scan IO.vi Block Diagram on Single-Board RIO PID Example.
File Edit View Project Operate Tools Window Help
13pt Application Font
Read FPGA I/O
Perform applicable scaling
Update Current Value Table using Shared Variable
FPGA VI Reference In
FPGA VI Reference Out
error In
Thermocouple 1
Mod1/CJC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

Figure 8.36. The RT Read FPGA Scan IO VI pulls all updates from the FPGA scan, performs applicable conversions and scaling, and publishes data to the memory 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 specialized features of one platform, such as the CompactRIO Scan Mode, the porting process is more involved, but, in that case, only the I/O portions of the code require change for migration. In both situations, all the LabVIEW processing and control algorithms are completely portable and reusable across RIO hardware platforms.