LabVIEW™

Real-Time Module User Manual

April 2004 Edition
Part Number 322154E-01

Worldwide Technical Support and Product Information

ni.com

National Instruments Corporate Headquarters

11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 683 0100

Worldwide Offices

Australia 1800 300 800, Austria 43 0 662 45 79 90 0, Belgium 32 0 2 757 00 20, Brazil 55 11 3262 3599,
Canada (Calgary) 403 274 9391, Canada (Ottawa) 613 233 5949, Canada (Québec) 450 510 3055,
Canada (Toronto) 905 785 0085, Canada (Vancouver) 514 685 7530, China 86 21 6555 7838,
Czech Republic 420 224 235 774, Denmark 45 45 76 26 00, Finland 385 0 9 725 725 11,
France 33 0 1 48 14 24 24, Germany 49 0 89 741 31 30, Greece 30 2 10 42 96 427, India 91 80 51190000,
Israel 972 0 3 6393737, Italy 39 02 413091, Japan 81 3 5472 2970, Korea 82 02 3451 3400,
Malaysia 603 9131 0918, Mexico 001 800 010 0793, Netherlands 31 0 348 433 466,
New Zealand 0800 553 322, Norway 47 0 66 90 76 60, Poland 48 22 3390150, Portugal 351 210 311 210,
Russia 7 095 783 68 51, Singapore 65 6226 5886, Slovenia 386 3 425 4200, South Africa 27 0 11 805 8197,
Spain 34 91 640 0085, Sweden 46 0 8 587 895 00, Switzerland 41 56 200 51 51, Taiwan 886 2 2528 7227,
Thailand 662 992 7519, United Kingdom 44 0 1635 523545

For further support information, refer to the *Technical Support and Professional Services* appendix. To comment on the documentation, send email to techpubs@ni.com.

Important Information

Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this document is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Trademarks

DataSocket™, FieldPoint™, LabVIEW™, National Instruments™, NI™, ni.com™, NI Developer Zone™, NI-CAN™, NI-DAQ™, NI-VISA™, and TestStand™ are trademarks of National Instruments Corporation.

Product and company names mentioned herein are trademarks or trade names of their respective companies.

Patents

For patents covering National Instruments products, refer to the appropriate location: **Help»Patents** in your software, the `patents.txt` file on your CD, or `ni.com/patents`.

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

(1) NATIONAL INSTRUMENTS PRODUCTS ARE NOT DESIGNED WITH COMPONENTS AND TESTING FOR A LEVEL OF RELIABILITY SUITABLE FOR USE IN OR IN CONNECTION WITH SURGICAL IMPLANTS OR AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT SYSTEMS WHOSE FAILURE TO PERFORM CAN REASONABLY BE EXPECTED TO CAUSE SIGNIFICANT INJURY TO A HUMAN.

(2) IN ANY APPLICATION, INCLUDING THE ABOVE, RELIABILITY OF OPERATION OF THE SOFTWARE PRODUCTS CAN BE IMPAIRED BY ADVERSE FACTORS, INCLUDING BUT NOT LIMITED TO FLUCTUATIONS IN ELECTRICAL POWER SUPPLY, COMPUTER HARDWARE MALFUNCTIONS, COMPUTER OPERATING SYSTEM SOFTWARE FITNESS, FITNESS OF COMPILERS AND DEVELOPMENT SOFTWARE USED TO DEVELOP AN APPLICATION, INSTALLATION ERRORS, SOFTWARE AND HARDWARE COMPATIBILITY PROBLEMS, MALFUNCTIONS OR FAILURES OF ELECTRONIC MONITORING OR CONTROL DEVICES, TRANSIENT FAILURES OF ELECTRONIC SYSTEMS (HARDWARE AND/OR SOFTWARE), UNANTICIPATED USES OR MISUSES, OR ERRORS ON THE PART OF THE USER OR APPLICATIONS DESIGNER (ADVERSE FACTORS SUCH AS THESE ARE HEREAFTER COLLECTIVELY TERMED "SYSTEM FAILURES"). ANY APPLICATION WHERE A SYSTEM FAILURE WOULD CREATE A RISK OF HARM TO PROPERTY OR PERSONS (INCLUDING THE RISK OF BODILY INJURY AND DEATH) SHOULD NOT BE RELIANT SOLELY UPON ONE FORM OF ELECTRONIC SYSTEM DUE TO THE RISK OF SYSTEM FAILURE. TO AVOID DAMAGE, INJURY, OR DEATH, THE USER OR APPLICATION DESIGNER MUST TAKE REASONABLY PRUDENT STEPS TO PROTECT AGAINST SYSTEM FAILURES, INCLUDING BUT NOT LIMITED TO BACK-UP OR SHUT DOWN MECHANISMS. BECAUSE EACH END-USER SYSTEM IS CUSTOMIZED AND DIFFERS FROM NATIONAL INSTRUMENTS' TESTING PLATFORMS AND BECAUSE A USER OR APPLICATION DESIGNER MAY USE NATIONAL INSTRUMENTS PRODUCTS IN COMBINATION WITH OTHER PRODUCTS IN A MANNER NOT EVALUATED OR CONTEMPLATED BY NATIONAL INSTRUMENTS, THE USER OR APPLICATION DESIGNER IS ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY OF NATIONAL INSTRUMENTS PRODUCTS WHENEVER NATIONAL INSTRUMENTS PRODUCTS ARE INCORPORATED IN A SYSTEM OR APPLICATION, INCLUDING, WITHOUT LIMITATION, THE APPROPRIATE DESIGN, PROCESS AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

Contents

About This Manual

This manual contains information about the LabVIEW Real-Time Module and real-time programming techniques to help you build a deterministic application.

Conventions

The following conventions appear in this manual:

»	The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box.
	This icon denotes a note, which alerts you to important information.
bold	Bold text denotes items that you must select or click in the software, such as menu items and dialog box options. Bold text also denotes parameter names.
italic	Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font also denotes text that is a placeholder for a word or value that you must supply.
`monospace`	Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames, and extensions.
(Platform)	Text in this font denotes a specific platform and indicates that the text following it applies only to that platform. The **(RTX)** platform tag denotes content specific to the LabVIEW Real-Time Module for RTX Targets. The **(ETS)** platform tag denotes content specific to the LabVIEW Real-Time Module for ETS Targets.

Related Documentation

The following documents contain information that you might find helpful as you read this manual:

- RT Series hardware documentation
- *Getting Started with the LabVIEW Real-Time Module*
- **(Mac OS)** *LabVIEW Real-Time Module for Mac OS X User Manual Addendum*, which describes the differences between the Windows and Mac OS X versions of the LabVIEW Real-Time Module
- *LabVIEW Real-Time Module Release Notes*
- *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**
- *Getting Started with LabVIEW*
- *LabVIEW User Manual*
- *LabVIEW Application Builder User Guide*

1

Introduction to the LabVIEW Real-Time Module

Most LabVIEW applications run on a general-purpose operating system (OS) like Windows, Linux, Solaris, or Mac OS. Some applications require deterministic real-time performance that general-purpose operating systems cannot guarantee. The LabVIEW Real-Time Module extends the capabilities of LabVIEW to address the need for deterministic real-time performance.

The Real-Time Module combines LabVIEW graphical programming with the power of a real-time operating system, enabling you to build deterministic real-time applications. You develop VIs in LabVIEW and execute the VIs on an RT target. The RT target runs VIs without a user interface and offers a stable platform for real-time VIs.

LabVIEW Real-Time Module Platforms

National Instruments designed the LabVIEW Real-Time Module to execute VIs on two different real-time platforms. The LabVIEW Real-Time Module can execute VIs on hardware targets running the real-time operating system of the Venturcom Phar Lap Embedded Tool Suite (ETS) and on computers running the Venturcom Real-Time Extension (RTX).

Venturcom Phar Lap ETS provides a real-time operating system that runs on NI RT Series hardware to meet the requirements of embedded applications that need to behave deterministically or have extended reliability requirements.

Venturcom RTX adds a real-time subsystem (RTSS) to Windows. Venturcom RTX enables you to run Windows and the RTSS at the same time on the same computer. The RTSS has a priority based real-time execution system independent of the Windows scheduler. RTSS scheduling supersedes Windows scheduling to ensure deterministic real-time performance of applications running in the RTSS. For more information about Phar Lap ETS or RTX refer to the Venturcom Web site

at www.vci.com. Refer to the *LabVIEW Real-Time Module Release Notes* for installation instructions.

Real-Time System Components

A real-time system consists of software and hardware components. The software components include LabVIEW, the RT Engine, and VIs you build using LabVIEW. The hardware components of a real-time system include a host computer and an RT target. The following sections describe the different components of a real-time system.

Host Computer

The host computer is a computer with LabVIEW and the LabVIEW Real-Time Module installed on which you develop the VIs for the real-time system. After developing the real-time system VIs, you can download and run the VIs on the RT target. The host computer can run VIs that communicate with the VIs running on the RT target.

LabVIEW

You develop VIs with LabVIEW on the host computer. The Real-Time Module extends the capabilities of LabVIEW to allow you to select an RT target on which to run VIs.

RT Engine

The RT Engine is a version of LabVIEW that runs on the RT target. The RT Engine runs the VIs you download to RT targets. The RT Engine provides deterministic real-time performance for the following reasons:

- The RT Engine runs on a real-time operating system (RTOS) or RTX subsystem, which ensures that the LabVIEW execution system and other services adhere to real-time operation. Refer to Chapter 3, *Building Deterministic Applications*, for information about the LabVIEW execution system.
- The RT Engine runs on RT Series hardware or the RT target on the RTX subsystem. Other applications or device drivers commonly found on the host computer do not run on RT targets. The absence of additional applications or devices means that a third-party application or driver does not impede the execution of VIs.
- RT targets on which the RT Engine runs do not use virtual memory, which eliminates a major source of unpredictability in deterministic systems.

RT Target

An RT target refers to RT Series hardware or the RTSS that runs the RT Engine and VIs you create using LabVIEW. There are three types of RT targets: RT Series plug-in devices, networked RT Series devices, and the RTSS.

RT Series Plug-In Devices (ETS Only)

An NI PCI-7041 plug-in device is a plug-in board with an embedded processor. The NI PCI-7041 plug-in device contains a processor board and data acquisition daughterboard. The processor board contains a microprocessor with a real-time operating system that runs the RT Engine and LabVIEW VIs. You can use the host computer to communicate with and control VIs running on the NI PCI-7041 plug-in device through an Ethernet connection or shared memory.

This manual does not contain information about the data acquisition daughterboard of plug-in devices. Refer to the appropriate plug-in device documentation for information about the data acquisition daughterboard for plug-in devices.

Networked RT Series Devices (ETS Only)

A networked RT Series device is a networked hardware platform with an embedded processor with a real-time operating system that runs the RT Engine and LabVIEW VIs. You can use a separate host computer to communicate with and control VIs on a networked RT Series device through an Ethernet connection, but the device is an independent computer. Some examples of networked RT Series devices include the following:

- NI RT Series PXI Controller—A networked device installed in an NI PXI chassis that communicates with NI PXI modules installed in the chassis. You can write VIs that use all the input/output (I/O) capabilities of the PXI modules, SCXI modules, and other signal conditioning devices installed in a PXI chassis. The RT Engine also supports features of the RT Series PXI controller. Refer to the National Instruments Web site at `ni.com/info` and enter the info code `RT0001` for information about the features supported by the RT Engine on specific networked devices.
- NI RT Series FieldPoint Module—A networked device ideal for distributed real-time I/O applications.

- NI 1450 Series Compact Vision System—An easy-to-use, distributed, real-time imaging system that acquires, processes, and displays images from IEEE 1394 cameras. Refer to the *NI 1450 Series Compact Vision User Manual* for information about using the 1450 Series Compact Vision System with LabVIEW.

This manual does not contain hardware-related information about specific networked devices. Refer to the appropriate device documentation for information about the device.

RT Target on the Real-Time Subsystem (RTX Only)

When you use the LabVIEW Real-Time Module for RTX Targets, the RT Engine runs on the RTSS of the host computer. The RTSS creates the *x*:\RTXROOT directory, where *x* is the Windows root drive, to store all of the RT target files. Like RT Series hardware targets, the RTSS provides a real-time platform where you can execute LabVIEW VIs deterministically. You can communicate with and control VIs running on the RTSS from LabVIEW in Windows.

VIs running on the RTSS can use an NI PCI-7831 plug-in device for data acquisition. Refer to Chapter 4, *Timing Applications and Acquiring Data*, for information about deterministic data acquisition using LabVIEW.

Real-Time Module and Express VI Considerations

LabVIEW Express VIs increase LabVIEW ease of use and improve productivity with interactive dialog boxes that minimize programming for measurement applications. Express VIs require additional performance overhead during execution, therefore do not use Express VIs in time-critical or processor-intensive applications. Instead, develop real-time applications with standard LabVIEW VIs. Refer to the *Getting Started with LabVIEW* manual for information about LabVIEW Express VIs.

LabVIEW shows an Express VI-oriented palette view by default. Complete the following steps to switch to the Advanced LabVIEW palette view.

1. Select **Tools»Options** from LabVIEW.
2. Select **Controls/Functions Palettes** from the **Options** dialog box pull-down menu.
3. Select **Advanced** from the **Palette View** pull-down menu.
4. Click the **OK** button.

Unsupported LabVIEW Features

RT targets do not support some LabVIEW features for VIs that run on the target. If you attempt to download and run on an RT target a VI that has unsupported functionality, the VI might still execute. However, the unsupported functions do not work and return standard LabVIEW error codes.

Modifying Front Panel Objects of RT Target VIs

When a VI or stand-alone application runs on an RT target, you cannot execute VIs that modify a front panel. For example, you cannot change or read the properties of front panel objects with property nodes because there is no front panel for VIs that run on the RT target. The VI still runs on the RT target but the front panel object is not affected and returns an error. In some cases, you can establish a front panel connection with the RT target or open a remote panel connection to read any front panel properties or reflect front panel property changes. Refer to Chapter 2, *Connecting to RT Targets*, for information about establishing a front panel connection with an RT target. Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, for information about establishing a remote panel connection to an RT target.

The following features only work on an RT target with a front panel connection:

- Front panel property nodes and control references
- Dialog VIs and functions
- VI Server front panel functions

The following features do not work on an RT target and return an error:

- Menu functions
- Cursor VIs
- The **Clear indicators when called** component of the **Execution Properties** page. Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, for information about the **Execution Properties** page.

Note Some LabVIEW example VIs use unsupported VIs and functions and do not execute on an RT target.

Using OS-Specific Technologies in RT Target VIs

VIs on the RT target cannot use VIs that leverage Windows-only technology. The following features do not work on an RT target:

- ActiveX VIs
- .NET VIs
- Windows Registry Access VIs
- TestStand VIs (ActiveX-based)
- Report Generation Toolkit VIs
- Report Express VI (Uses Report Generation Toolkit VIs)
- Graphics and Sound VIs
- Database Connectivity Toolset
- XML DOM Parser and G Web Server for CGI Support
- **(ETS)** Call Library Nodes that access an operating system API other than Venturcom Phar Lap ETS
- **(RTX)** Call Library Nodes that access an operating system API other than Venturcom RTX
- **(RTX)** LabVIEW Timed Loop

2

Connecting to RT Targets

When you first launch LabVIEW after installing the Real-Time Module, the default execution target is the host computer operating system, as shown in Figure 2-1. Refer to the *LabVIEW Real-Time Module Release Notes* for installation instructions.

Figure 2-1. LabVIEW Dialog Box

You can select an RT target or the host computer as the LabVIEW execution target. When you select an execution target other than the host computer, LabVIEW establishes a front panel connection with the RT target and downloads any LabVIEW VIs you subsequently run to the selected execution target.

Complete the following steps to select an RT target as the execution target.

1. Start LabVIEW.

2. Select the execution target from the **Execution Target** pull-down menu.

   (RTX) Select **RT Target: RTX (RTRTX::0)** from the **Execution Target** pull-down menu.

Complete the following steps to select a networked RT Series device as the execution target if you have not selected the device previously.

1. Select **Select Target with Options** from the **Execution Target** pull-down menu to open the **Select Execution Target** dialog box shown in Figure 2-2.
2. Select **RT Target on Network** from the pull-down menu.

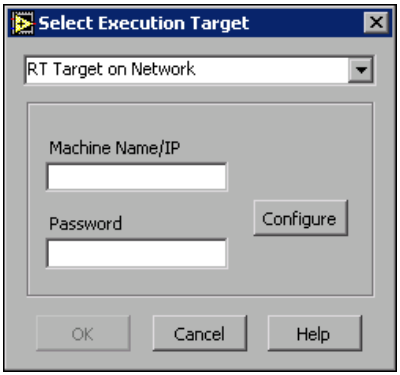

Figure 2-2. Select Execution Target Dialog Box

3. Enter the IP address of the RT target in the **Machine Name/IP** text box.
4. Enter the password of the RT target in the **Password** text box. Leave the **Password** text box blank if the RT target does not have a password set. You can set the password of an RT target using Measurement & Automation Explorer (MAX).

Note If you have not configured the hardware in MAX, click the **Configure** button to open MAX. Refer to the *Remote Systems Help*, by selecting **Help»Help Topics»Remote Systems** from MAX, for information about using MAX to configure RT Series hardware.

Press the <F5> key to refresh the pull-down menu of the **Select Execution Target** dialog box.

5. Click the **OK** button.

Downloading VIs to an RT Target

When you select an RT target in the **Select Execution Target** dialog box, LabVIEW establishes a front panel connection with the RT target. You can download a VI and its associated subVIs to an RT target by clicking the **Run** button. The RT Engine on the RT target then runs the downloaded VI.

Note When you edit a VI or convert a VI from a different version of LabVIEW, you must save the VI on the host computer before you can download and run it on the RT target.

You also can download LabVIEW VIs without running them by selecting **Operate»Download Application** after selecting an RT target as the execution target. You can determine which VIs are downloaded to the RT target by selecting **Browse»Show VI Hierarchy** to open the **Hierarchy** window. The VI hierarchy appears with a pin in the upper left corner of each VI.

When the pin is in the vertical position, as shown to the left, the VI downloaded to the RT target.

When the pin is in the horizontal position, as shown to the left, the VI has not downloaded to the RT target.

Closing a Front Panel Connection without Closing VIs

You also can exit LabVIEW on the host computer without closing the VIs on the RT target. Select **File»Exit without closing RT Engine VIs** to close LabVIEW on the host computer. The VIs running on the RT target continue running. VIs downloaded but not running remain loaded in memory on the RT target.

If you select **File»Exit**, LabVIEW opens a dialog box that asks if you want to exit LabVIEW without closing RT Engine VIs. If you click the **Yes** button, LabVIEW exits without closing the VIs on the RT target. If you click the **Close all RT Engine VIs** button, LabVIEW closes all the VIs running on the RT target, unloads the VIs from memory, and closes LabVIEW.

You also can select **Operate»Switch Execution Target** and then select another execution target. LabVIEW opens a new front panel connection to the RT target you select. The VIs running on the original RT target continue to run.

Connecting to VIs Running on an RT Target

When you connect LabVIEW to an RT target to open a front panel connection, LabVIEW detects VIs currently running on the RT target. LabVIEW attempts to open the local copy of the VIs to show the front panel.

Note When connecting to an RT Series plug-in device, if you place a checkmark in the **Reset** checkbox on the **Select Execution Target** dialog box, LabVIEW clears all VIs in memory on the target.

If you moved or modified the local copies of the VIs since you downloaded them to the RT target, LabVIEW displays the **Changed or Missing VIs** dialog box, shown in Figure 2-3.

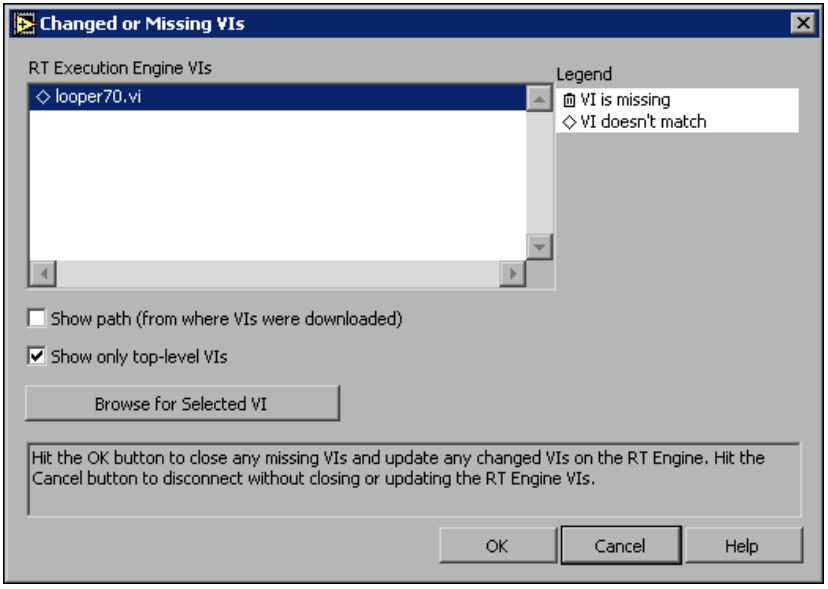

Figure 2-3. Changed or Missing VIs Dialog Box

The **Changed or Missing VIs** dialog box shows the name of the local VIs that are missing or that have been modified and no longer match the VIs running on the RT target. Use the **Changed or Missing VIs** dialog box to browse for a VI you moved on the host computer file system, to close all VIs running on the RT target and update them with the latest version of each VI, and to close the front panel connection between LabVIEW and the RT target while leaving the VIs running.

Configuring RT Target Options

You can set access permissions and start-up options for RT targets. With the RT target selected as the execution target, select **Tools»RT Target: *X* Options** to access the RT Target **Options** dialog box, where ***X*** is the IP address of a networked RT target or the device name of a plug-in device.

(RTX) Select **Tools»RT Target: RTX (RTRTX::0) Options** to access the RT Target **Options** dialog box.

Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, for information about the RT Target **Options** dialog box.

3

Building Deterministic Applications

Determinism is the characteristic of a system that describes how consistently it responds to external events or performs operations within a given time limit. If you intend to build deterministic applications, use the programming techniques in this chapter to achieve high levels of determinism in VIs.

Most computers have only one processor, so tasks execute one at a time. Computers achieve multitasking by running one application for a short amount of time and then running other applications for short amounts of time. As long as the amount of processor time allocated for each application is small enough, computers appear to have multiple applications running simultaneously.

Multithreading is when you apply the concept of multitasking to a single application by breaking it into smaller tasks that execute for short amounts of time in different execution system threads. A *thread* is a completely independent flow of execution for an application within the execution system. Multithreaded applications maximize the efficiency of the processor because the processor does not sit idle if there are other threads ready to run. Any application that reads and writes from a file, performs I/O, or polls the user interface for activity can benefit from multithreading because you can use the processor to run other tasks during breaks in these activities.

Creating Multithreaded Applications in LabVIEW

To create a multithreaded application in LabVIEW, you must separate time-critical tasks from non-time-critical tasks. You then can build VIs to complete each task. You prioritize the VIs and then categorize them into one of the available execution systems to control the amount of processor resources each VI receives. LabVIEW assigns each VI to an execution

system thread according to the VI priority and execution system you assign. The threads execute on the processor accordingly. You can use deterministic communication methods to pass data between the different VIs.

The real-time operating system (RTOS) on RT targets and the RTSS use a combination of round robin and preemptive scheduling to execute threads in the execution system. Round robin scheduling applies to threads of equal priority. Equal shares of processor time are allocated among equal priority threads. For example, each normal priority thread is allotted 10 ms to run. The processor executes all the tasks it can in 10 ms and whatever is incomplete at the end of that period must wait to complete during the next allocation of time. Conversely, preemptive scheduling means that any higher priority thread that needs to execute immediately pauses execution of all lower priority threads and begins to execute. A time-critical priority thread is the highest priority and preempts all priorities.

Dividing Tasks to Create Deterministic Multithreaded Applications

Deterministic control applications depend on time-critical tasks to complete on time, every time. Therefore, time-critical tasks need enough processor resources to ensure their completion. Separate time-critical tasks from all other tasks in the application and place them in a separate VI so you can ensure they receive enough processor resources. For example, if a control application processes measurement data at regular intervals and stores the data on disk, you must handle the timing and control of the data acquisition in a time-critical VI. However, storing the data on disk is inherently a non-deterministic task because file I/O operations have unpredictable response times that depend on the hardware and the availability of the hardware resource. Place file I/O operations in a normal priority VI.

The time-critical priority VI receives the processor resources necessary to complete the task and does not relinquish control of the processor until it cooperatively yields to the normal priority VI or until it completes the task. The normal priority VI then runs until preempted by the time-critical VI. You can use deterministic methods to pass data between the VIs running on the RT target. Refer to the *Passing Data between VIs* section of this chapter for information about deterministic communication methods you can use to pass data between the time-critical VI and lower priority VIs on the RT target.

If the application contains two normal priority VIs in addition to the time-critical VI, the timing of the application changes. For example, if the above application also requires updates to a VI running on a host computer,

you must create a separate normal priority VI for network communication. The network communication VI can receive data from other VIs on the RT target using deterministic communication methods. The communication VI then can execute the non-deterministic network communication code to pass data to the VI running on the host computer. When the application runs, the time-critical VI uses the processor resources until the task completes or until it cooperatively relinquishes control. The normal priority VI that logs data and the normal priority VI that performs the network communication with the host computer round robin the control of the processor resources in equal amounts of time until the tasks complete or until preempted by the time-critical VI again for control of the processor resources.

After separating all deterministic tasks from non-deterministic tasks in the application into different VIs, assign priorities and an execution system to the VIs. You then can time the VIs to cooperatively yield execution to allow other lower priority VIs to run.

Note (RTX) VIs that monopolize processor resources prevent Windows processes from executing. Windows returns an RTX Starvation Timeout error. You must allow Windows processes enough processor resources to run by including sleep in RT target VIs. Refer to the NI Web site at `ni.com/info` and enter the info code `RTX001` for information about RTX Starvation Timeout errors that might occur when using the LabVIEW Real-Time Module for RTX Targets.

Assigning Priorities to VIs

You can select from the following VI priorities, listed in order from lowest to highest, to assign VIs to an execution system thread:

- background priority (lowest)
- normal priority
- above normal priority
- high priority
- time-critical priority (highest)

Threads of higher priority preempt threads of lower priority.
Normal priority is the default thread priority for all VIs you create in LabVIEW. The time-critical priority preempts all thread priorities.
A time-critical priority thread does not relinquish processor resources until it completes all tasks. However, a time-critical thread can explicitly relinquish control of processor resources to ensure that the thread does not monopolize the processor resources. SubVIs inherit the priority of the

caller VI. For example, a subVI called in a time-critical VI runs in time-critical priority.

Note Because time-critical priority threads cannot preempt each other, create only one time-critical thread in an application to guarantee deterministic behavior.

In addition to the five priority levels listed above, you can set VIs to subroutine priority. VIs set for subroutine priority do not share execution time with other VIs. When a VI runs at the subroutine priority level, it effectively takes control of the thread in which it is running, and it runs in the same thread as its caller. No other VI can run in that thread until the subroutine VI finishes running, even if the other VI is at the subroutine priority level.

Complete the following steps to set the priority of a VI.

1. Select **File»VI Properties** to open the **VI Properties** dialog box.
2. Select **Execution** from the **Category** pull-down menu.
3. Select the priority from the **Priority** pull-down menu.

Assigning VIs to Execution Systems

LabVIEW has the following six execution systems to help you categorize VIs:

- user interface
- standard
- instrument I/O
- data acquisition
- other 1
- other 2

The names of the execution systems are suggestions for the type of VIs to place within the execution system. By default, all VIs run in the standard execution system at normal priority. The user interface execution system handles all user interface tasks. You can assign instrument I/O and data acquisition task VIs to other execution systems, but the labels help to organize the VIs. In addition to the six execution systems, you also can assign VIs to the *same as caller* execution system. The same as caller category runs subVIs in the same execution system as the VI that called the subVI.

Every execution system except user interface has a thread queue. For example, if you have three threads assigned to an execution system, at any time, one thread might run as the other two wait in the queue. Assuming all threads have the same priority, one thread runs for a certain amount of time. The thread then moves to the end of the queue, and the next thread runs. When a thread completes, the execution system removes the thread from the queue.

The execution systems are not responsible for managing the user interface. If a thread in one queue needs to update the user interface, the execution system passes responsibility to the user interface execution system, which updates the user interface.

Complete the following steps to set the execution system of a VI.

1. Select **File»VI Properties** to open the **VI Properties** dialog box.
2. Select **Execution** from the **Category** pull-down menu.
3. Select the execution system from the **Preferred Execution System** pull-down menu.

Cooperatively Yielding Time-Critical VI Execution

Because of the preemptive nature of time-critical VIs, they can monopolize processor resources. A time-critical VI might use all of the processor resources, not allowing lower priority VIs in the application to execute.

You must build time-critical VIs that periodically yield, or sleep, to allow lower priority tasks to execute without affecting the determinism of the time-critical code. By timing control loops, you can yield time-critical VIs and cooperatively relinquish processor resources. Refer to Chapter 4, *Timing Applications and Acquiring Data*, for information about the methods available for timing time-critical VIs to relinquish processor resources.

(RTX) The execution system of the RTSS supersedes Windows scheduling. For this reason, VIs that monopolize processor resources prevent Windows processes from executing. Windows returns an RTX Starvation Timeout error. You must allow Windows processes enough processor resources to run by including sleep in RT target VIs. Refer to the NI Web site at `ni.com/info` and enter the info code `RTX001` for information about RTX Starvation Timeout errors that might occur when using the LabVIEW Real-Time Module for RTX Targets.

Passing Data between VIs

After dividing tasks in an application into separate VIs of varying priorities, you might need to communicate between the different VIs on the RT target. Use global variables, functional global variables, and the Real-Time FIFO VIs to send and receive data between VIs in an application.

Global Variables

Use global variables to access and pass small amounts of data between VIs, such as from a time-critical VI to a lower priority VI.

Global variables are a lossy form of communication, meaning the global variable can overwrite the data before you read the data. Tasks in a lower priority VI might not have enough processor time to read the data before other tasks in a different VI overwrite the data.

A global variable is a shared resource that you must use carefully in a time-critical VI. If you use a global variable to pass data out of a time-critical VI, you must ensure that a lower priority VI reads the data and unlocks the global before the time-critical VI attempts to write to the global again. Refer to Chapter 5, *Optimizing Applications*, for information about shared resources.

Using a global variable is a good way to pass data smaller than 32-bits, such as scalar data, between VIs. For larger amounts of data, use functional global variables or the Real-Time FIFO VIs. Refer to the *LabVIEW User Manual* for information about creating and using global variables.

Functional Global Variables

Use functional global variables to pass data between VIs. A functional global variable is a subVI set to subroutine priority. The subVI contains a While Loop with a nested Case structure for read and write access. Figure 3-1 shows the read and write cases of the Case structure for a functional global variable. The While Loop contains uninitialized shift registers that store data. A functional global variable receives an action input that specifies which task the VI performs, as shown in Figure 3-1 by the **Mode** input parameter. Any subsequent calls to the functional global variable can access the most recent data. Functional global variables resemble queues because you can add more shift registers to store a longer history of values. You also can add more than one set of shift registers to pass more than one set of data.

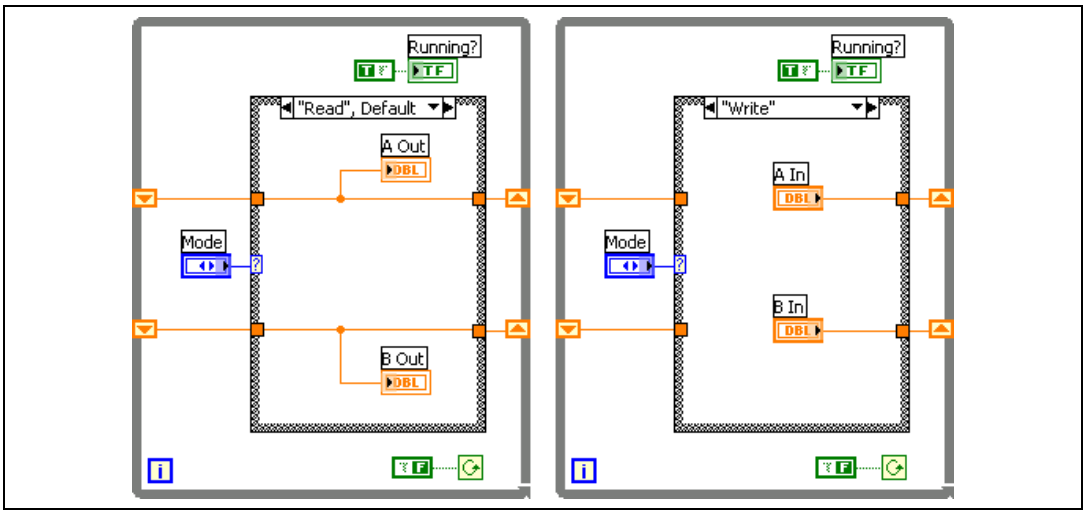

Figure 3-1. Read and Write Case of a Functional Global Variable

Unlike global variables, you can implement functional global variables such that they are not a shared resource. If you right-click on a subVI set to subroutine priority and select **Skip Subroutine Call If Busy** from the shortcut menu, the execution system skips the subVI if the subVI is currently running in another thread. Skipping a subVI helps in time-critical VIs because the VI does not wait for the subVI. If you skip the execution of a subVI, the subVI returns the default value for that data type and not the default indicator value. For example, the default data type value for numerics is zero, strings and arrays default to an empty string, and Booleans default to FALSE. If you want to detect the execution of a functional global variable, wire a TRUE constant to a Boolean output on the functional global variable block diagram, as shown in Figure 3-1. If the Boolean output returns a TRUE value, the functional global variable executed. If the Boolean output returns the default value of FALSE, the functional global variable did not execute. Skip functional global variables in time-critical VIs but not in lower priority VIs. In lower priority VIs, you can wait to receive non-default values.

Functional global variables can be a lossy form of communication if a VI overwrites the shift register data before another VI reads the data.

Refer to the `examples\Real-Time\RT Communication.llb` for examples of using functional global variables to communicate between VIs that run on an RT target.

Real-Time FIFO VIs

Use the Real-Time FIFO VIs to transfer data between VIs in an application. An RT FIFO acts like a fixed queue, where the first value you write to the FIFO is the first value that you can read from the FIFO. RT FIFOs and LabVIEW queues both transfer data from one VI to another. However, unlike a LabVIEW queue, an RT FIFO ensures deterministic behavior by imposing a size restriction on the data. You must define the number and size of the RT FIFO elements. Both a reader and writer can access the data in an RT FIFO at the same time, allowing RT FIFOs to work safely from within a time-critical VI.

Use the RTFIFOCreate VI to create a new FIFO or open a reference to a FIFO that you created in another VI. Use the RTFIFORead and RTFIFOWrite VIs to read and write data to the FIFO. Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, for VI reference information about the Real-Time FIFO VIs and for information about the data types supported by the Real-Time FIFO VIs.

Because of the fixed-size restriction, an RT FIFO can be a lossy communication method. Writing data to an RT FIFO when the FIFO is full overwrites the oldest element. You must read data stored in an RT FIFO before the FIFO is full to ensure the transfer of every element without losing data. Check the **overwrite** output of the RTFIFOWrite VI to ensure that you did not overwrite data. If the RT FIFO overwrites data, the **overwrite** output returns a TRUE value. Refer to the `examples\Real-Time\RT Communication.llb` for examples of using the Real-Time FIFO VIs to communicate between VIs that run on an RT target.

Communicating with Applications on an RT Target

The RT Engine on the RT target does not provide a user interface for applications. You can use one of two communication protocols, front panel communication or network communication, to provide a user interface on the host computer for RT target VIs.

Front Panel Communication

With front panel communication, LabVIEW and the RT Engine execute different parts of the same VI, as shown in Figure 3-2. LabVIEW on the host computer displays the front panel of the VI while the RT Engine executes the block diagram. A user interface thread handles the communication between LabVIEW and the RT Engine.

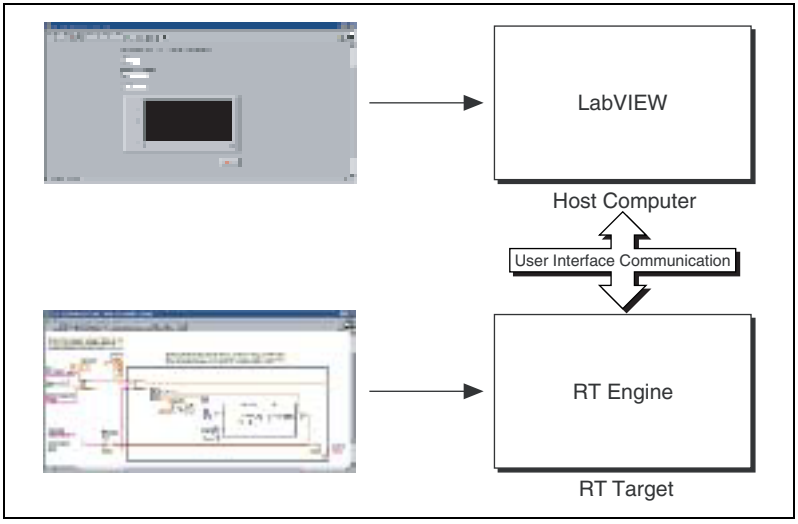

Figure 3-2. Front Panel Communication Protocol

Use front panel communication between LabVIEW on the host computer and the RT Engine to control and test VIs running on an RT target. After downloading and running the VIs, keep LabVIEW on the host computer open to display and interact with the front panel of the VI.

You also can use front panel communication to debug VIs while they run on the RT target. You can use LabVIEW debugging tools—such as probes, execution highlighting, breakpoints, and single stepping—to locate errors on the block diagram code. Refer to Chapter 6, *Debugging Deterministic Applications*, for information about debugging applications.

Front panel communication is a good communication method to use during development because front panel communication is a quick method for monitoring and interfacing with VIs running on an RT target. However, front panel communication is not deterministic and can affect the determinism of a time-critical VI. Use network communication methods to increase the efficiency of the communication between a host computer and VIs running on the RT target.

Network Communication

With network communication, a host VI runs on the host computer and communicates with the VI running on the RT target using specific network communication methods such as TCP, VI Server, and in the case of non-networked RT Series plug-in devices, shared memory reads and writes. You might use network communication for the following reasons:

- You want to run another VI on the host computer.
- You want to control the data exchanged between the host computer and the RT target. You can customize communication code to specify which front panel objects get updated and when. You also can control which components are visible on the front panel because some controls and indicators might be more important than others.
- You want to control timing and sequencing of the data transfer.
- You want to perform additional data processing or logging.

In Figure 3-3, the RT target VI is similar to the VI in Figure 3-2 that runs on the RT target using front panel communication to update the front panel controls and indicators. However, the RT target VI in Figure 3-3 uses Real-Time FIFO VIs to pass data to a communication VI. The communication VI then communicates with the host computer VI using network communication methods to update controls and indicators. Refer to the *Passing Data between VIs* section of this chapter for information about the methods available to send data between VIs running on an RT target. Refer to the *Exploring Communication Methods* section of this chapter for information about methods available to send data between VIs running on an RT target and VIs running on the host computer.

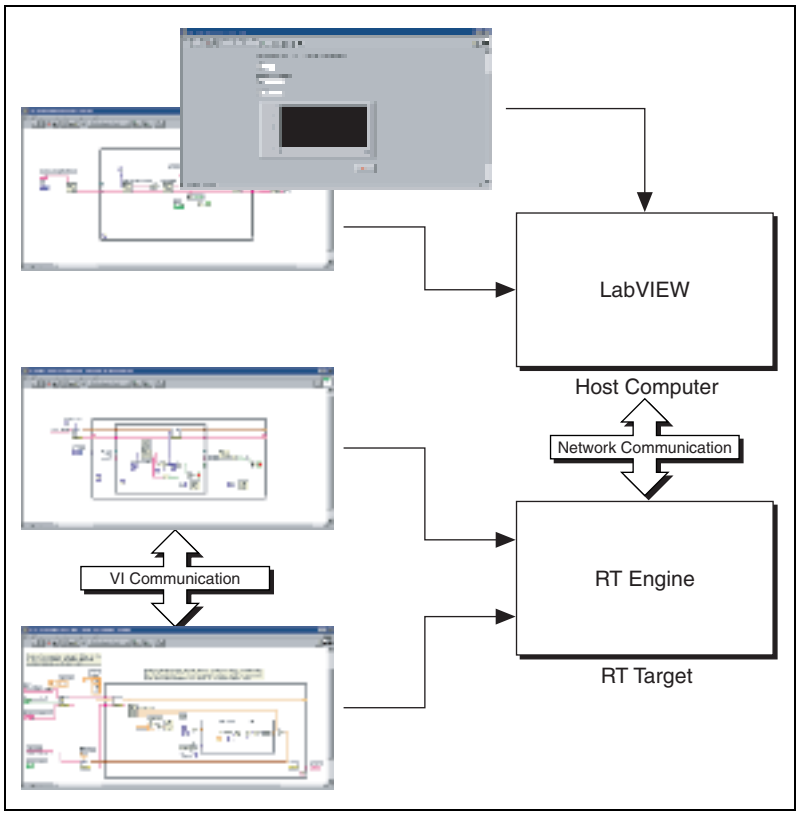

Figure 3-3. Network Communication Protocol

Creating Communication VIs with the RT Communication Wizard

Use the RT Communication Wizard to create VIs that deterministically transfer front panel control and indicator data from time-critical VIs running on an RT target to a VI running on the host computer.

Select **Tools»RT Communication Wizard** to open the RT Communication Wizard and specify a time-critical VI. The RT Communication Wizard returns a list of controls and indicators present in the time-critical VI you specify. The RT Communication Wizard replaces the front panel controls and indicators you select from the time-critical VI with Real-Time FIFO VIs. The RT Communication Wizard creates a normal priority VI that contains Real-Time FIFO VIs to send and receive

front panel data from the time-critical VI. The Real-Time FIFO VIs transfer data deterministically and do not affect the timing of the time-critical VI.

The RT Communication Wizard creates the following three VIs:

- **Time-Critical VI**—Runs on the RT target and contains the time-critical tasks and Real-Time FIFO VIs to transfer front panel data deterministically to the normal priority VI.
- **Normal Priority VI**—Runs on the RT target and contains all non-deterministic network communication tasks to update the host VI with front panel data received from the time-critical VI.
- **Host VI**—Runs on the host computer and displays the front panel controls and indicators of the time-critical VI.

The VIs generated by the RT Communication Wizard transfer data between the RT target and the host computer using a single communication method. If you want more than one communication method to handle the front panel communication code or you want to vary the order in which the network communication VI sends and receives data, use network communication methods to manually create a network communication VI. Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, for information about using the RT Communication Wizard to create communication VIs to display the front panel of a time-critical VI on the host computer. Refer to the *Exploring Communication Methods* section of this chapter for information about the network communication methods you can use in LabVIEW.

Exploring Communication Methods

You can use high-level software protocols to communicate between VIs running on the RT target and VIs running on a host computer. Each protocol has its advantages and disadvantages. The following list classifies the different communication methods:

- Shared Memory Communication—Used for communication between LabVIEW and RT Series plug-in devices or the RT target on the RTX subsystem.
- Network Communication—Used for communication over Ethernet networks.
  - TCP
  - UDP
  - DataSocket

  - VI Server
  - SMTP (send only)
- Bus Communication—Used for communication over different bus communication ports.
  - Serial
  - CAN
- IrDA Wireless Communication—Used for communication with RT targets using IrDA hardware.

Shared Memory

In operating systems like Windows, two processes or applications can communicate with each other using the shared memory mechanism of the operating system. Similarly, VIs running on an RT target and VIs running on the host computer can communicate using the Real-Time Shared Memory VIs. Use the Real-Time Shared Memory VIs to read and write to shared memory locations of RT Series plug-in devices or the shared memory locations of the real-time subsystem.

The Real-Time Shared Memory VIs communicate data deterministically because they have low overhead. However, the NI RT Series PCI-7041 plug-in devices have a shared memory size limit of 512 KB. If you need to transfer several megabytes of data, you must divide the data into smaller portions and then transfer them. In doing so, you must make sure you do not overwrite the data in the shared memory before it is read. Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, for information about the Real-Time Shared Memory VIs.

(RTX) Use `RTRTX::0` as the device name of the RT target when communicating with the target using the Real-Time Shared Memory VIs.

Network Communication

Table 3-1 lists the characteristics of the different network communication methods.

Table 3-1. Characteristics of Network Communication Protocols

Protocol	Speed	Reliability
TCP	Fast	Lossless
UDP	Very Fast	Lossy
DataSocket	Fast	Lossy
VI Server	Slow	Lossless*
SMTP	Fast	Lossless**

* Not for large data transfers. Used for monitoring or controlling one shot runs of remote VIs. Primarily suited for remote control of VIs.

** You can only send data out from the RT target.

TCP

TCP is an industry-standard protocol for communicating over networks. VIs running on the host computer can communicate with RT target VIs using the LabVIEW TCP functions. However, TCP is non-deterministic, and using TCP communication inside a time-critical VI might cause the loop cycle time to vary from the desired time. Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, for information about the LabVIEW TCP functions.

The Real-Time Module extends the capabilities of the existing TCP functions to enable communication with networked RT Series devices and to allow communication across shared memory with RT Series plug-in devices.

(RTX) Use `RTRTX::0` as the address of an RT target when attempting to open a TCP connection with the target using the LabVIEW TCP functions.

UDP

UDP is a network transmission protocol for transferring data between two locations on a network. UDP is not a connection-based protocol, so the transmitting and receiving computers do not establish a network connection. Because there is no network connection, there is little overhead when transmitting data. However, UDP is non-deterministic, and using

UDP communication inside a time-critical VI might cause the loop cycle time to vary from the desired time.

When using UDP to send data, the receiving computer must have a read port open before the transmitting computer sends the data. Use the UDP Open function to open a write port and specify the IP address and port of the receiving computer. The data transfer occurs in byte streams of varying lengths called datagrams. Datagrams arrive at the listening port and the receiving computer buffers and then reads the data.

You can transfer data bidirectionally with UDP. With bidirectional data transfers, both computers specify a read and write port and transmit data back and forth using the specified ports. You can use bidirectional UDP data transfers to send and receive data from the network communication VI on the RT target.

UDP has the ability to perform fast data transmissions deterministically. However, UDP cannot guarantee that all datagrams arrive at the receiving computer. Because UDP is not connection based, you cannot verify the arrival of datagrams. You must ensure that network congestion does not affect the transmission of datagrams. Also, you must read data stored in the data buffer of the receiving computer fast enough to prevent overflow and loss of data. Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, for information about using the UDP functions to exchange data with devices on a remote UDP port.

(RTX) You must use `RTRTX::0` as the address of the RT target when attempting to write data to the target using the LabVIEW UDP functions.

DataSocket (ETS Only)

DataSocket is an Internet programming technology to share live data between VIs and other computers. A DataSocket Server running on a host computer acts as a data repository. Data placed on the DataSocket Server becomes available for clients to access.

Note You can bind a DataSocket connection to a front panel object and control the object from a client connection. However, this feature is not supported by VIs on an RT target because there is no front panel.

One advantage of using DataSocket is that multiple clients can access data on the DataSocket Server. A LabVIEW VI can use the DataSocket Write VI to post data to the DataSocket Server. Any number of VIs running on different RT targets or host computers can use the DataSocket Read VI to

retrieve the data. RT target VIs can post data, such as status information, to the DataSocket Server for a VI running on a host computer to read.

DataSocket is non-deterministic and using DataSocket functions inside a time-critical VI adds jitter to the application. Refer to the *LabVIEW User Manual* for information about DataSocket technology. Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, for information about the LabVIEW DataSocket VIs and functions.

VI Server

Use the VI Server to monitor and control VIs on an RT target. Using VI Server technology, a LabVIEW VI can invoke RT target VIs remotely. The LabVIEW VI can pass parameter values to and from the RT target VIs, creating a distributed application.

A host VI can invoke only VIs in memory on the RT target. The host VI cannot dynamically download LabVIEW VIs to the RT target for use with the VI Server. Refer to the *Downloading VIs to an RT Target* section of Chapter 2, *Connecting to RT Targets*, for information about downloading VIs without running them.

One advantage to communicating using the VI Server is that the VI Server allows you to access the functionality of TCP while working within the framework of LabVIEW. However, the VI Server is non-deterministic and using VI Server communication inside a time-critical VI adds jitter to the application. Refer to the *LabVIEW User Manual* for information about using the VI Server.

(RTX) You must use `RTRTX::0` as the computer name of the RT target when attempting to open an application reference using the VI Server.

SMTP (ETS Only)

Use the SMTP VIs to send data from a VI running on the RT target to VIs running on another computer. The SMTP VIs can send electronic mail, including attached data and files, using the Simple Mail Transfer Protocol (SMTP). You cannot use the SMTP VIs to receive information.

SMTP is non-deterministic, and using SMTP communication inside a time-critical VI adds jitter to the application. Refer to the *LabVIEW User Manual* for information about emailing data from a VI.

Bus Communication

Serial (ETS Only)

Serial communication is the transmission of data between two locations through the serial ports. The VISA functions provide serial communication support in LabVIEW for communication between RT targets with serial devices and serial instruments or computers that have a serial connection. Serial communication is ideal when transfer data rates are low or for transmitting data over long distances. You must install NI-Serial RT on the RT target from MAX. **(Mac OS)** Install NI-Serial RT using the Remote System Explorer.

Serial communication is non-deterministic, and using serial communication inside a time-critical VI adds jitter to the application. Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, for information about using the LabVIEW VISA functions for serial communication.

CAN (ETS Only)

Controller Area Network (CAN) is a deterministic, multi-drop communication bus standardized as ISO 11898. Using CAN, you can transfer up to 8 data bytes per frame at a rate of up to 1 Mbit per second. You can network multiple RT systems using NI-CAN interface cards and NI-CAN driver software. You cannot use CAN communication with RT Series plug-in devices. You must install NI-CAN RT on the RT target from MAX. **(Mac OS)** Install NI-CAN RT using the Remote System Explorer.

Refer to the *NI-CAN Hardware and Software Manual* for information about using NI-CAN hardware and software with LabVIEW.

IrDA Wireless Communication (ETS Only)

Infrared Data Association (IrDA) is a communication standard that specifies a way to transfer data using a wireless infrared connection. IrDA devices communicate using infrared LEDs. You can use IrDA devices to send data in and out of VIs running on an RT target using the LabVIEW IrDA functions. RT Series controllers support Extended Systems XTNDAccess IrDA PC Adapters and ACTiSYS IR-220L+ IrDA Com-Port Serial Adapters connected to a built-in controller serial port. You must install NI-IrDA RT on the RT target from MAX.

IrDA is non-deterministic, and using IrDA communication inside a time-critical VI adds jitter to the application. Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, for information about using the LabVIEW IrDA functions for IrDA communication and for information about configuring IrDA on an RT target.

4

Timing Applications and Acquiring Data

This chapter explains how to time applications to periodically yield execution of the time-critical VI and allow lower priority VIs to execute.

Timing Control Loops

Because of the preemptive nature of the RTOS on RT Series devices, a time-critical application thread can monopolize the processor on the device. A thread might use all processor resources and not allow lower priority threads in the application to execute. The time-critical task must periodically yield processor resources to the lower-priority tasks so they can execute. By properly separating the time-critical task from lower priority tasks, you can reduce application jitter. Refer to Chapter 3, *Building Deterministic Applications*, for information about building deterministic VIs that reduce application jitter.

You can use software methods or a hardware method to time control loops. The software method is available for RT Series FieldPoint modules, RT Series PXI controllers, and RT Series plug-in devices. The hardware method is available only for RT Series plug-in devices and RT Series PXI controllers.

Timing Control Loops Using Software

LabVIEW provides two functions, Wait (ms) and Wait Until Next ms Multiple, to time loops with sleep modes of 1 kHz. If you need to achieve faster loop rates, use hardware timing. Refer to the *Timing Control Loops Using Hardware* section of this chapter for information about using NI hardware to achieve smaller loop rates.

Wait (ms)

The Wait (ms) function causes a VI to sleep for the specified amount of time. For example, if the operating system millisecond timer value is 112 ms when the Wait (ms) function executes, and the **milliseconds to**

wait input equals 10, then the function returns when the millisecond timer value equals 122 ms.

Avoid using this VI in parallel with anything in a time-critical priority. If the Wait (ms) function executes first, the whole thread sleeps until the Wait (ms) finishes, and the code in parallel does not execute until the Wait (ms) finishes. The resulting loop period is the code execution time plus the **milliseconds to wait** time.

Wait Until Next ms Multiple Function

The Wait Until Next ms Multiple function causes a thread to sleep until the operating system millisecond timer value equals a multiple of the **millisecond multiple** input. For example, if the Wait Until Next ms Multiple function executes with a **millisecond multiple** input of 10 ms and the operating system millisecond timer value is 112 ms, the VI sleeps until the millisecond timer value equals 120 ms because 120 ms is the first multiple of 10 ms after the Wait Until Next ms Multiple function executes.

Use the Wait Until Next ms Multiple function to synchronize a loop with the operating system millisecond timer value multiple. A loop has a period of **millisecond multiple** if the Wait Until Next ms Multiple function executes in parallel with other code in the same loop. However, the loop does not have the period of **millisecond multiple** if the code takes longer to execute than the **millisecond multiple**.

Avoid placing the Wait Until Next ms Multiple function in parallel with other code because doing so can result in incorrect timing of a control system. Instead, use a sequence structure to control when the Wait Until Next ms Multiple function executes. Figure 4-1 shows the ideal timing of a control system. The example control system reads an analog input, writes an analog output, and waits in a synchronized cycle that repeats.

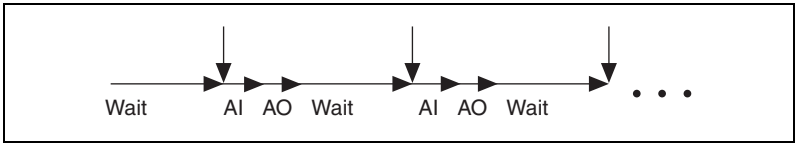

Figure 4-1. Ideal Timing of a Control System

The block diagram in Figure 4-2 uses a While Loop with the Wait Until Next ms Multiple function to create a control system.

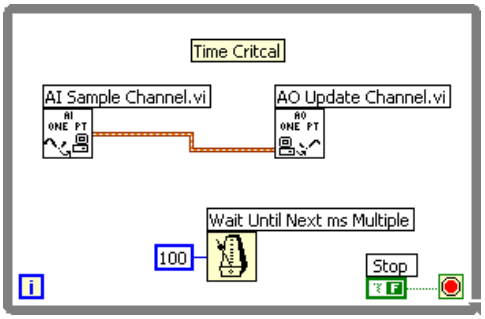

Figure 4-2. Parallel Implementation of a Control System

Because of the dataflow properties of LabVIEW programming, the Wait Until Next ms Multiple function can execute before, after, or between the execution of the analog input and output. The behavior of the loop differs depending on when the Wait Until Next ms Multiple function executes. If the Wait Until Next ms Multiple function executes first, the analog input precedes the analog output. The control system timing matches the ideal timing shown in Figure 4-1. However, when the Wait Until Next ms Multiple function does not execute first, the loop sleeps until the function executes. Because a portion of the loop sleeps, the entire loop sleeps, and the analog output waits until the loop returns from sleep to execute, which produces results that do not match the ideal timing. Figure 4-3 shows the incorrect results.

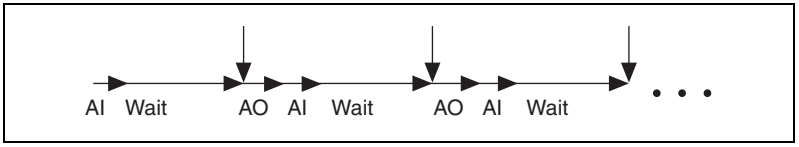

Figure 4-3. Incorrect Timing of a Control System

You can use a Flat Sequence structure to force a specific execution sequence, as shown in Figure 4-4. In the block diagram in Figure 4-4, the Wait Until Next ms Multiple function precedes the analog input and the system implementation produces results that match the ideal timing.

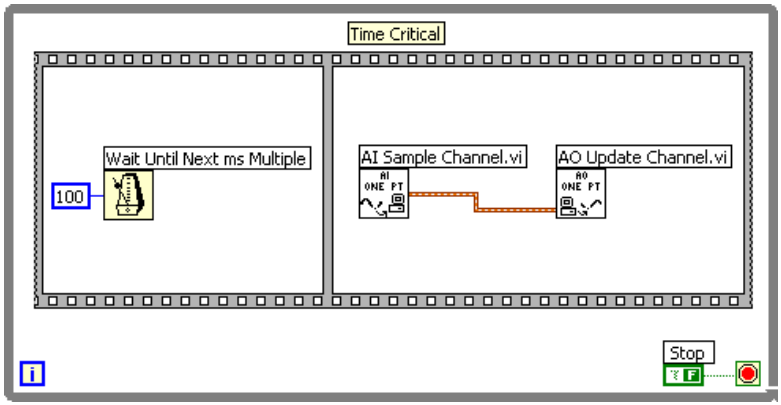

Figure 4-4. Flat Sequence Structure Implementation of a Control System

Real-Time Timing VIs

You can use the Real-Time Timing VIs to time control loops. Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, for VI reference information about the Real-Time Timing VIs.

LabVIEW Timed Loop (ETS Only)

The LabVIEW Timed Loop executes each iteration of a loop at the period you specify. The Timed Loop can specify the period for each iteration of the loop by setting a periodic alarm. In addition to specifying the period, you can specify the offset, the timing source, and the priority of the loop. Refer to the *Using the Timed Loop to Write Multi-Rate Applications in LabVIEW* Application Note for information about the Timed Loop.

Timing Control Loops Using Hardware

You can use NI data acquisition hardware and NI-DAQmx to achieve a sleep resolution much finer than 1 kHz. Refer to the *NI-DAQmx Help* for more information about timing control loops.

Acquiring Data with VIs Running on an RT Target

You can create VIs that run on an RT target and acquire data deterministically using National Instruments hardware.

RT Series Data Acquisition Devices (ETS Only)

To perform data acquisition using NI-DAQmx supported hardware, refer to the *NI-DAQmx Help* for information about configuring and programming the hardware for acquiring data using NI-DAQmx 7.2.

RT Series FieldPoint Modules (ETS Only)

To perform data acquisition using FieldPoint I/O modules, refer to the *Measurement & Automation Explorer Help for FieldPoint*, available by selecting **Help»Help Topics»FieldPoint**, for information about configuring the hardware and for information about programming the hardware for acquiring data using the FieldPoint VIs and LabVIEW.

NI PCI-7831 Plug-in Device (ETS and RTX)

The LabVIEW Real-Time Module supports the NI PCI-7831 plug-in device for data acquisition. To perform data acquisition using the NI PCI-7831 plug-in device and the LabVIEW Real-Time Module, refer to the *Getting Started with the NI 7831R* for information about configuring and programming the hardware for acquiring data.

5

Optimizing Applications

This chapter explains techniques that can improve the determinism of applications.

Avoiding Shared Resources

In LabVIEW, there are resources that two or more VIs might need to share. These shared resources include global variables, non-reentrant subVIs, the LabVIEW Memory Manager, queues, semaphores, single-threaded DLLs, and so on. If a VI uses a shared resource, the VI acquires an operating system mutex around the resource to protect the resource from access by other VIs. A *mutex* locks the resource so that other VIs may not access the resource. If a time-critical priority VI preempts a lower priority VI and attempts to use a locked resource, the time-critical VI must wait because the shared resource is not available. The lower priority VI becomes more important than the time-critical VI because it must finish its work and release the shared resource before the time-critical VI can proceed. This scenario is a *priority inversion*.

Priority inversions induce jitter. Avoid or minimize jitter in a time-critical VI to ensure determinism. Do not use shared resources in time-critical VIs. The amount of jitter induced by a shared resource depends on the type of shared resource involved. For instance, when accessing a global variable of fixed size, a VI can finish a read or write operation on the global variable within a consistent length of time or with very little variance in time. Because reading and writing to a fixed size global variable is bound in time, you can account for the jitter induced by sharing the global variable.

Memory Allocations and Preallocating Arrays

When a VI allocates memory, the VI accesses the LabVIEW Memory Manager. The LabVIEW Memory Manager allocates memory for data storage. The LabVIEW Memory Manager is a shared resource and might be locked by a mutex up to several milliseconds. Avoid allocating memory within a time-critical VI control loop.

If you are using arrays in time-critical VI control loops, you can reduce jitter by preallocating arrays before entering the loop.

The block diagram in Figure 5-1 builds an array within the control loop. Jitter affects the loop because the Build Array function inside the loop uses the LabVIEW Memory Manager at every iteration to allocate memory for the array.

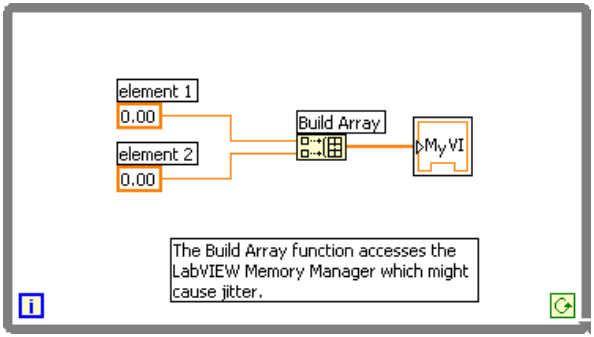

Figure 5-1. Memory Allocation in Control Loop

The block diagram in Figure 5-2 uses the Initialize Array function outside the loop and the Replace Array Subset function inside the loop to create the array. Because the array is preallocated outside the control loop, the control loop no longer needs to access the LabVIEW Memory Manager at every iteration.

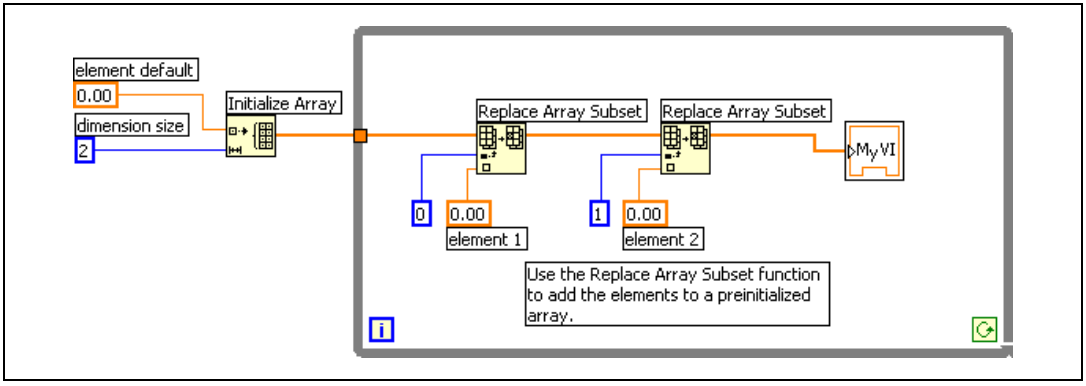

Figure 5-2. Preallocated Array

Casting Data to Proper Data Types

Cast data to the proper data type in VIs running on the RT target. Every time LabVIEW performs a type conversion, LabVIEW makes a copy of the data buffer in memory to retain the new data type after the conversion. The LabVIEW Memory Manager must allocate memory for the copy, which might affect the determinism of time-critical VIs. Also, creating copies of the data buffer takes up memory resources on an RT target. Refer to the *LabVIEW User Manual* for more information about casting data types.

Use the smallest data type possible when casting the data type. If you must convert the data type of an array, do the conversion before you build the array. Also, keep in mind that a function output reuses an input buffer only if the output and the input have the same data type. Arrays must have the same structure and number of elements for function outputs to reuse the input buffer.

Reducing the Use of Global Variables

LabVIEW creates an extra copy in memory of every global variable you use in a VI. Reduce the number of global variables to improve the efficiency and performance of VIs. Creating copies of the global variable takes up memory resources on an RT target.

Avoiding Contiguous Memory Conflicts

LabVIEW handles many of the memory details that you normally deal with in a conventional, text-based language. For example, functions that generate data must allocate storage for the data. When that data is no longer needed, LabVIEW deallocates the associated memory. When you add new information to an array or a string, LabVIEW allocates new memory to accommodate the new array or string. However, running out of memory is a concern with VIs running on an RT target.

You must design memory-conscious VIs for RT targets. Always preallocate space for arrays equal to the largest array size that you might encounter.

When you reboot or reset an RT target, the RTOS and the RT Engine load into memory as shown in diagram 1 of Figure 5-3.

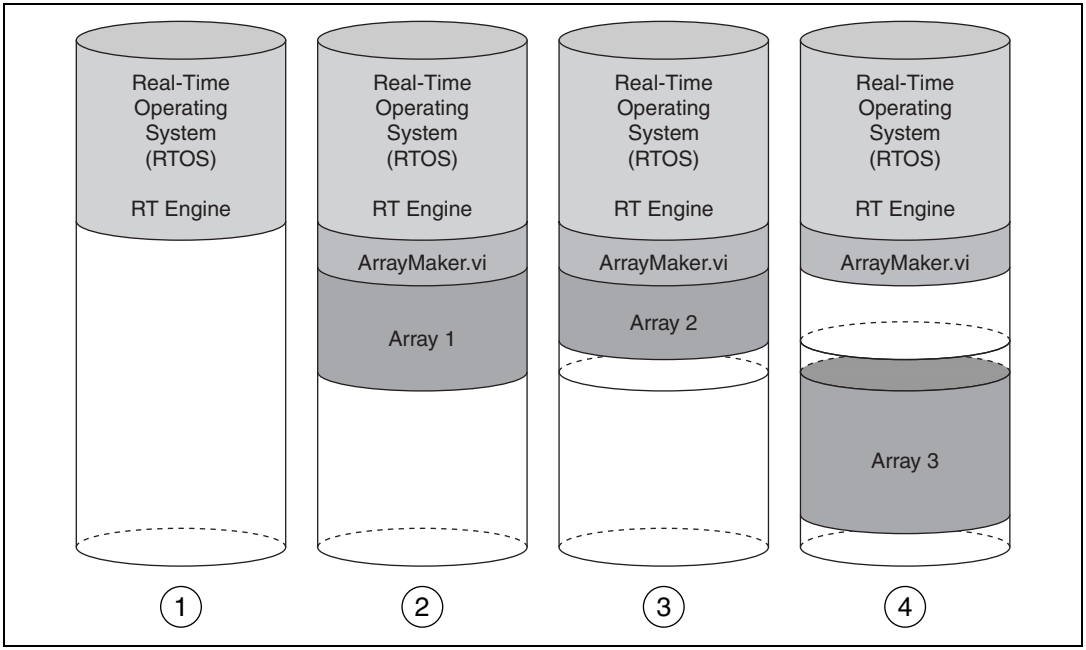

Figure 5-3. Memory Diagrams of an RT Target

The RT Engine uses available memory for running RT target VIs and storing data. In diagram 2 of Figure 5-3, `ArrayMaker.vi` creates Array 1. All elements in Array 1 must be contiguous in memory.

The RTOS reuses the same memory addresses if you stop a VI and then run it again with arrays of the same size or smaller. In diagram 3 of Figure 5-3, `ArrayMaker.vi` creates Array 2. The RTOS creates Array 2 in the reserved memory space previously occupied by Array 1. Array 2 is small enough to fit in the reserved memory space that was allocated to Array 1. The extra contiguous memory used for Array 1 remains in the reserved memory space, as shown in diagram 3 of Figure 5-3.

When `ArrayMaker.vi` runs for a third time with a larger array or if another VI generates a larger array, the RT Engine must find a large enough contiguous space. In diagram 4 of Figure 5-3, `ArrayMaker.vi` must create Array 3, larger than the previous arrays, in the available memory.

Even when `ArrayMaker.vi` stops running, the RT Engine continues to run. Previously reserved memory is not available. If `ArrayMaker.vi` runs a fourth time and attempts to create an array larger than Array 3, the operation fails. There is no contiguous memory area large enough to create

the array because of the memory fragmentation. You can preserve memory space by preallocating array space equal to the largest use case.

Avoiding SubVI Overhead

Calling a subVI from a VI running on an RT target adds a small amount of overhead to the overall application. Although the overhead is small, calling a subVI multiple times in a loop can add a significant amount of overhead. You can embed the loop in the subVI to reduce the overhead.

You also can convert subVIs into subroutines by changing the VI priority. The LabVIEW execution system minimizes the overhead to call subroutines. Subroutines are short, frequently executed tasks that generally do not require user interaction. Subroutines cannot display front panel data and do not multitask with other VIs. Also, avoid using timing or dialog box functions in subroutines. Refer to Chapter 3, *Building Deterministic Applications*, for information about setting VI priorities.

Setting VI Properties

To reduce memory requirements and increase performance of VIs, disable nonessential options in the **VI Properties** dialog box available by selecting **File»VI Properties**. Select **Execution** from the **Category** pull-down menu and remove checkmarks from the **Allow debugging** and **Auto handle menus at launch** checkboxes. By disabling these options, VIs use less memory, compile quicker, and perform better overall.

Mass Compiling VIs

Mass compiling a VI is another way to improve performance. Mass compiling a VI compiles the top-level VI and also recompiles and re-links all the subVIs and functions on the block diagram with the top-level VI.

Minimizing Memory Usage by the RT Target Web Server

To minimize memory usage when you enable the Web Server on an RT target, include only the VIs you want to access remotely on the **Web Server: Visible VIs Options** page of the RT Target **Options** dialog box. Also, disable the Web Server or remove all VIs on the **Web Server: Visible VIs Options** page to avoid losing memory when the Web Server is not in use.

Setting BIOS Options (ETS Only)

You can improve performance of VIs running on RT Series PXI controllers if you disable USB hardware in the BIOS of the controller. Make sure the PXI controller has the following BIOS setting:

```
Integrated Peripherals
    USB Keyboard = DISABLED
```

Additionally, for NI PXI-8170 controllers, make sure the controller has the following BIOS setting:

```
PnP/PCI Configuration
    Assign IRQ for USB = DISABLED
```

6

Debugging Deterministic Applications

Building deterministic LabVIEW applications requires programming techniques different from typical LabVIEW programming. With deterministic applications, there are two levels of debugging to verify—application behavior and timing behavior.

Verifying Correct Application Behavior

You first must ensure that the application behaves as expected. Use the LabVIEW debugging tools to detect errors and step through the flow of execution to locate the error source. Finally, use the **Profile** window or the LabVIEW Execution Trace Toolkit to test the execution timing and memory usage of an application.

Using the LabVIEW Debugging Tools

Use the LabVIEW debugging tools, such as execution highlighting and single-stepping, while the host computer is connected to an RT target to step through LabVIEW code.

Note You must place a checkmark in the **Allow debugging** checkbox of the **Execution** page of the **VI Properties** dialog box to use the LabVIEW debugging tools to debug a VI.

The only feature not supported by the Real-Time Module is the Call Chain ring, which appears in the toolbar of a subVI block diagram window while single-stepping. Refer to the *LabVIEW User Manual* for information about the LabVIEW debugging tools.

Note Do not use the LabVIEW debugging tools to debug execution timing because all debugging tools affect the timing of an application.

Using the Profile Window

The **Profile** window is a powerful tool for statistically analyzing how an application uses execution time and memory. The **Profile** window displays information that can identify the specific VIs or parts of VIs you need to optimize. For example, if you notice that a particular subVI takes a long time to execute, you can improve the performance of that VI. The **Profile** window displays the performance information for all VIs in memory in an interactive tabular format. From the **Profile** window, you can select the type of information to gather and sort the information by category. You also can monitor subVI performance within different VIs. Select **Tools»Advanced»Profile VIs** to display the **Profile** window.

Note You must connect LabVIEW to an RT target while running the **Profile** window.

You must place a checkmark in the **Profile Memory Usage** checkbox before starting a profiling session. Collecting information about VI memory use adds a significant amount of overhead to VI execution, which affects the accuracy of any timing statistics gathered during the profiling session. Therefore, perform memory profiling separate from time profiling to return an accurate profile.

Many of the options in the **Profile** window become available only after you begin a profiling session. During a profiling session, you can take a snapshot of the available data and save it to an ASCII spreadsheet file. The timing measurements accumulate each time you run a VI. Refer to the *LabVIEW Performance and Memory Management* Application Note for information about using the **Profile** window.

Using the Real-Time System Manager (ETS Only)

The Real-Time System Manager displays details about VIs running on an RT target and provides a dynamic display of the performance of the target. You also can stop VIs and start idle VIs using the Real-Time System Manager. Select **Tools»Real-Time System Manager** to open the Real-Time System Manager. Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, for information about using the Real-Time System Manager.

Verifying Correct Timing Behavior

Timing is crucial in a deterministic application. Use one of the following methods to verify the timing of an application.

Using the Tick Count (ms) Function

Use the Tick Count (ms) function to measure the time it takes to perform N iterations of a specified operation and calculate the average time in seconds per operation or average operations per second. Refer to the Benchmarking Shell VI located in the `examples/Real-Time/RT Tutorial.llb` for an example using the Tick Count (ms) function to benchmark code. Refer to the NI Developer Zone at `ni.com/zone` for information about using the Tick Count (ms) function to verify timing behavior.

Using the NI Time Stamp VIs

On Pentium processor-based RT targets, you can obtain a timestamp using the NI Timestamp VIs. The NI Timestamp VIs, located in the `vi.lib/addons/rt/_RTUtility.llb`, read the Time Stamp Counter register from the Pentium processor of RT targets for every loop iteration. Refer to the NI Timestamp Code Timer VI located in the `examples/Real-Time/RT Timing.llb` for an example using the NI Timestamp VIs to benchmark code. Refer to the National Instruments Web site at `ni.com/info` and enter the info code `NItstamp f`or information about the using the NI Timestamp VIs to verify timing behavior.

Using an Oscilloscope

The Tick Count (ms) function and the NI Timestamp VIs allow you to examine the relative timing of sections of LabVIEW VIs and to measure the software jitter in the VIs. Sometimes you might need to measure the jitter of the whole system at the hardware level, especially when you use hardware timing and the software jitter is masked out at the system level. Use external tools such as an oscilloscope to study the relationship between input and output signals and to measure loop rates and jitter levels.

Using Software Drivers

Sometimes you can use software drivers to make sure that the application is capable of keeping up with the desired loop rate. For example, you can use the DAQmx Is Read or Write Late VI to determine whether a control loop is keeping real time. Refer to Developer Zone at `ni.com/zone f`or information about using DAQmx VIs to time control loops.

Using the LabVIEW Execution Trace Toolkit (ETS Only)

The LabVIEW Execution Trace Toolkit is a real-time event and execution tracing tool that allows you to capture and display the timing and event data of VI and thread events for LabVIEW Real-Time Module applications in Windows. Refer to the National Instruments Web site at `ni.com/info` and enter the info code `lvtrace` for information about the LabVIEW Execution Trace Toolkit.

Using and Defining Error Codes

Use error handling to debug and manage errors in VIs. The LabVIEW error handler VIs return error codes when an error occurs in a VI. Error codes reveal the specific problem the VI encountered. When you configure an RT target, LabVIEW automatically copies the error code files used by the error handler VIs to the target.

You can use custom error codes with VIs that run on an RT target. Create error files using the Error Code File Editor by selecting **Tools»Advanced»Edit Error Codes**. If you use custom errors with LabVIEW, you must rename the files to use a `*.err` extension and then place the error files in the `c:\ni-rt\system\user.lib\errors` directory or the `c:\ni-rt\system\errors` directory on the RT target. Use the FTP client in MAX or any other FTP client to transfer the error file to the networked device. Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, for information about defining custom error codes.

7

Deploying Applications

Use the LabVIEW Application Builder, included with the Real-Time Module Professional Development System, to create stand-alone Real-Time Module applications. Stand-alone applications do not require you to run them from within a LabVIEW environment. You can embed a stand-alone application on an RT target and launch the application automatically when you boot the target.

Building Stand-Alone Applications

In LabVIEW, select **Tools»Build Application or Shared Library (DLL)** to launch the LabVIEW Application Builder. Refer to the *LabVIEW Application Builder User Guide* for information about using the Application Builder to build a stand-alone application and then refer to this chapter for information about building stand-alone applications for RT targets.

Note You must select an RT target as the execution target before launching the Application Builder if you want to embed the stand-alone application on the target.

Configuring Target Settings

On the **Target** tab, the Application Builder determines the target file name, destination directory, and support file directory from the **Application Path** text box in the RT Target **Options** dialog box. Select **Tools»RT Target: *x.x.x.x* Options**, where ***x.x.x.x*** is the IP address of the RT target, and then select **RT Target: Miscellaneous** from the pull-down menu to open the **RT Target: Miscellaneous** page.

(RTX) Select **Tools»RT Target: RTX (RTRTX::0) Options** and then select **RT Target: Miscellaneous** from the pull-down menu to open the **RT Target: Miscellaneous** page.

You cannot change the application name, destination directory, or support file directory settings from the **Target** tab because the Application Builder determines the path from the **Application Path** text box in the RT Target **Options** dialog box.

If you place a checkmark in the **Small target file with external file for subVIs** checkbox of the **Build Options** section, you cannot change the **LLB for other files** path because Application Builder determines the path from the **Application Path** text box in the RT Target **Options** dialog box. Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, for information about the RT Target **Options** dialog box.

Saving Stand-Alone Applications

The Application Builder saves stand-alone applications on the host computer or embeds applications on an RT target.

You must select an RT target as the execution target before launching the Application Builder if you want to embed the stand-alone application on the target. Refer to the *Launching Stand-Alone Applications* section of this chapter for information about launching embedded stand-alone applications on the RT target when the target boots up.

You also can launch the application on the host computer and then select an RT target as the execution target. Refer to the *Launching Applications Automatically Using Command Line Arguments* section of this chapter for information about automatically launching stand-alone applications using command line arguments.

Selecting a Target after Launch

When you run a stand-alone application built with the Application Builder on the host computer, the **Select Execution Target** dialog box appears. You can select any execution target for the application. Remove the checkmark from the **Show LabVIEW Real-Time target selection dialog when launched** checkbox in the **Application Settings** tab of the Application Builder to run the application on the host computer. You also can use command line arguments to launch the applications and specify an execution target. Refer to the *Launching Applications Automatically Using Command Line Arguments* section of this chapter for information about command line arguments.

Quitting LabVIEW after Launch

If you design a stand-alone application to run on an RT target, you might want the host computer to disconnect from the RT target after you download and run the application. Place a checkmark in the **Quit LabVIEW Real-Time host application after downloading** checkbox in the **Application Settings** tab of the Application Builder to have LabVIEW disconnect after launching the application on the RT target. Placing a checkmark in the **Quit LabVIEW Real-Time host application**

after downloading checkbox is equivalent to selecting **File»Exit without closing RT Engine VIs** in LabVIEW after downloading and running a VI on an RT target.

RT targets have media storage devices where you can save stand-alone applications. For example, the RT Series PXI controller has a hard disk drive, and the FieldPoint 20*xx* network module has flash. You can automatically launch embedded stand-alone applications each time you start the RT target. Refer to the *Launching Stand-Alone Applications* section of this chapter for information about launching embedded stand-alone applications automatically on an RT target.

Creating an Application Installer

In addition to building applications, the Application Builder can create an installer for a stand-alone application. You then can distribute the installer to other computers. The installer copies the stand-alone application and the necessary support files to the computer. Select the **Installer Settings** tab in the **Build Application or Shared Library (DLL)** dialog box and place a checkmark in the **Create installer** checkbox to access the installer options.

Note You must select the host computer as the execution target before launching the Application Builder if you want to create an installer for a stand-alone application.

Click the **Advanced** button to open the **Advanced Installer Settings** dialog box. Place checkmarks in the **LabVIEW Run-Time Engine** and the **LabVIEW RT Support** checkboxes to add the LabVIEW Run-Time Engine and support for RT Series hardware to the application installer.

You must include the LabVIEW Run-Time Engine if you plan to install the application on a computer that does not have the Real-Time Module installed.

Note **(RTX)** You must install Venturcom RTX and the LabVIEW Real-Time Module for RTX on the computer on which you want to install the application. You also must install the NI-VISA 3.1 driver from the National Instruments Device Driver CD and NI-RIO 1.1 for R Series devices from the NI-RIO for R Series Devices CD if you want to use the NI PCI-7831 plug-in device for instrument I/O.

The LabVIEW Run-Time Engine allows you to select an execution target to run the application.

When you install on a computer a stand-alone application that connects to an RT Series PXI controller, RT Series FieldPoint module, or NI PCI-7041 plug-in device, you also must install the following components from the National Instruments Device Driver CD on the computer:

- MAX 3.1
- PCI-7041 support
- FieldPoint 4.1 support (for FieldPoint)
- NI-DAQmx 7.2 or Traditional NI-DAQ 7.1
- Any other necessary I/O hardware drivers

Note RT targets must already have a version of the RT Engine installed that matches the version of LabVIEW you use to build the stand-alone application.

Launching Stand-Alone Applications

You can embed the application directly on an RT target and then launch it automatically on startup. You also can launch the application from the host computer and select an execution target on startup using command line arguments.

Launching Embedded Applications Automatically

The RT Engine on an RT target can launch embedded stand-alone applications each time you boot the target. **(RTX)** The embedded stand-alone application launches on the RTX subsystem when you log into Windows.

Complete the following steps to launch an embedded stand-alone application when the RT target boots up.

1. Select the RT target as the execution target.
2. Select **Tools»RT Target: *x.x.x.x* Options**, where ***x.x.x.x*** is the IP address of the device. **(RTX)** Select **Tools»RT Target: RTX (RTRTX::0) Options**.

   Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, for information about the RT Target **Options** dialog box.
3. Select **RT Target: Miscellaneous** from the pull-down menu.
4. Place a checkmark in the **Launch Application at Boot-up** checkbox.

Launching Applications Automatically Using Command Line Arguments

When you launch a stand-alone application on a host computer, the application launches the **Select Execution Target** dialog box. From the **Select Execution Target** dialog box, you can select the RT target on which you want the application to execute. You also can use command line arguments to disable the **Select Execution Target** dialog box and explicitly specify an RT target for the application. Use command line arguments in a batch file or shortcut from the host computer operating system startup folder to automatically launch stand-alone applications when the host computer starts.

To disable the **Select Execution Target** dialog box, specify the RT target in a command line argument using `-target`. For example,

```
c:\mybuiltapp_rtengine.exe -target RTRTX::0
```

You also can target the host computer and automatically launch a host computer application on startup. For example,

```
c:\mybuiltapp_host.exe -target host
```

To disconnect the host computer from the RT target after downloading the application, leaving the embedded application running, use `-quithost`. For example, the following command automatically downloads and runs `mybuiltapp_rtengine.exe` on an RT target with the IP address of `10.0.40.76` and then closes LabVIEW on the host computer.

```
c:\mybuiltapp_rtengine.exe -target 10.0.40.76 -quithost
```

Note If the host computer requires a login before entering the operating system, the application starts only after the login completes.

You also can reset a specified plug-in device using `-reset`. For example,

```
c:\mybuiltapp_rtengine.exe -target RT::0 -reset
```

This command automatically resets the plug-in device specified by `-target` before downloading the application. Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, for information about LabVIEW command line arguments.

Connecting to Stand-Alone Applications on an RT Target

You cannot open a front panel connection to a stand-alone application running on an RT target. Use a remote panel connection to connect to the application or use the RT Communication Wizard to create communication VIs to show the front panel of a time-critical VI on the host computer. Refer to the *LabVIEW Help* for information about using remote panels to connect to the front panel of a VI running on an RT target. Refer to Chapter 3, *Building Deterministic Applications*, for information about using the RT Communication Wizard.

A

Technical Support and Professional Services

Visit the following sections of the National Instruments Web site at ni.com for technical support and professional services:

- **Support**—Online technical support resources include the following:
  - **Self-Help Resources**—For immediate answers and solutions, visit our extensive library of technical support resources available in English, Japanese, and Spanish at ni.com/support. These resources are available for most products at no cost to registered users and include software drivers and updates, a KnowledgeBase, product manuals, step-by-step troubleshooting wizards, conformity documentation, example code, tutorials and application notes, instrument drivers, discussion forums, a measurement glossary, and so on.
  - **Assisted Support Options**—Contact NI engineers and other measurement and automation professionals by visiting ni.com/support. Our online system helps you define your question and connects you to the experts by phone, discussion forum, or email.
- **Training and Certification**—Visit ni.com/training for self-paced training, eLearning virtual classrooms, interactive CDs, and Certification program information. You also can register for instructor-led, hands-on courses at locations around the world.
- **System Integration**—If you have time constraints, limited in-house technical resources, or other project challenges, NI Alliance Program members can help. To learn more, call your local NI office or visit ni.com/alliance.

If you searched ni.com and could not find the answers you need, contact your local office or NI corporate headquarters. Phone numbers for our worldwide offices are listed at the front of this manual. You also can visit the Worldwide Offices section of ni.com/niglobal to access the branch office Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

Glossary

Symbol	Prefix	Value
m	milli	10^{-3}
k	kilo	10^{3}
M	mega	10^{6}
G	giga	10^{9}

A

address — Character code that identifies a specific location or series of locations in memory.

application — A collection of VIs that together accomplish the real-time system task.

D

DAQ — *See* data acquisition (DAQ).

data acquisition (DAQ) — (1) Acquiring and measuring analog or digital electrical signals from sensors, transducers, and test probes or fixtures; (2) Generating analog or digital electrical signals.

DataSocket — An Internet programming technology to share live data between VIs and other computers.

determinism — Characteristic of a system that describes how consistently it can respond to external events or perform operations within a given time limit.

device — An instrument or controller you can access as a single entity that controls or monitors real-world I/O points. A device often is connected to a host computer through some type of communication network.

driver — Software unique to the device or type of device, and includes the set of commands the device accepts.

E

embedded application	A stand-alone application built using the LabVIEW Application Builder and embedded as the start-up application on an RT target.
execution target	The target on which to run a LabVIEW VI. Can be either an RT target or the host computer.

F

FIFO	First-in-first-out memory buffer. The first data stored is the first data sent to the acceptor.
front panel communication	A protocol for communication between LabVIEW on the host computer and the RT Engine on an RT target, typically used during development.
functional global variable	A subVI set to subroutine priority used to pass data between several VIs on a block diagram.

G

global variable	A variable that accesses and passes data between several VIs on a block diagram.

H

host computer	The computer on which you develop VIs using LabVIEW and download them to an RT target.
Hz	Hertz—The number of scans read or updates written per second.

I

I/O	Input/Output—The transfer of data to/from a computer system involving communications channels, operator interface devices, and/or data acquisition and control interfaces.
INI	A file used to set application configuration options.

J

jitter	The amount of time that the loop cycle time varies from the desired time.

M

Measurement & Automation Explorer (MAX)	A graphical user interface for configuring National Instruments hardware and software.
media storage device	A device capable of storing data.
multitasking	When a computer runs applications for a short amount of time to give the impression of multiple applications running simultaneously.
multithreading	Running tasks of an application for a short amount of time to give the impression of multiple tasks running simultaneously.
mutex	An operating system lock around a resource to protect the resource from access by other VIs.

N

network communication	A protocol for communication between a host VI and a VI running on the RT target using specific network communication programmatic controls.
networked device	A hardware platform with an embedded processor that you can control using a separate host computer through an Ethernet connection.
NI-DAQmx	Driver software included with all NI measurement devices. NI-DAQmx is an extensive library of VIs and functions you can call from an application development environment (ADE), such as LabVIEW, to program all the features of an NI measurement device, such as configuring, acquiring and generating data from, and sending data to the device.

O

operating system	Base-level software that controls a computer, runs programs, interacts with users, and communicates with installed hardware or peripheral devices.

P

PCI	Peripheral Component Interconnect—A high-performance expansion bus architecture originally developed by Intel to replace ISA and EISA. It is achieving widespread acceptance as a standard for PCs and workstations; it offers a theoretical maximum transfer rate of 132 Mbytes/s.
plug-in device	Plug-in NI PCI/PXI RT Series devices with embedded processors. Each plug-in device contains a processor board and data acquisition daughterboard.
preemptive scheduling	Execution system scheduling of threads in which higher priority threads execute before all other threads.
priorities	Assigned to VIs to classify execution sequence in the execution system. Higher priority threads execute first, while threads with a lower priority wait.
Property Node	Sets or finds the properties of a VI or application.
Proportional Integral Derivative (PID) Control	Combination of proportional, integral, and derivative control actions. Refers to a control method in which the controller output is proportional to the error, its time history, and the rate at which it is changing. The error is the difference between the observed and desired values of a variable that is under control action.
protocol	The exact sequence of bits, characters, and control codes used to transfer data between computers and peripherals through a communications channel, such as the GPIB bus.
PXI	PCI eXtensions for Instrumentation. A modular, computer-based instrumentation platform.

R

real-time	A property of an event or system in which data is processed as it is acquired instead of being accumulated and processed at a later time.
Real-Time Operating System (RTOS)	Uses a combination of round robin and preemptive scheduling to execute threads in the execution system.
Real-Time Subsystem (RTSS)	A real-time subsystem in Windows that has a priority based real-time execution system independent of the Windows scheduler.
remote panel	Allows you to operate a front panel on a machine that is separate from where the VI resides and executes using a LabVIEW client or Web browser.
round robin scheduling	Scheduling that attempts to share the processor in equal amounts of time among all ready tasks of the same priority.
RT Engine	A version of LabVIEW that runs on the RT target.
RT target	RT Series hardware that runs VIs downloaded from and built in LabVIEW.
run-time engine	Runs LabVIEW executables built using the LabVIEW Application Builder on computers without LabVIEW.

S

shared memory	Memory that can be sequentially accessed by more than one controller or processor but not simultaneously accessed. Also known as dual-mode memory.
shift register	Optional mechanism in loop structures to pass the value of a variable from one iteration of a loop to a subsequent iteration. Shift registers are similar to static variables in text-based programming languages.
soft reboot	Restarting a computer without cycling the power, usually through the operating system.
subVI	VI used on the block diagram of another VI. Comparable to a subroutine.
synchronous	(1) Hardware—A property of an event that is synchronized to a reference clock; (2) Software—A property of a function that begins an operation and returns only when the operation is complete.

T

target	*See* RT target.
TCP	Transmission Control Protocol—A standard format for transmitting data in packets from one computer to another.
thread	A completely independent flow of control in an application.
Traditional NI-DAQ	An upgrade to the earlier version of NI-DAQ. Traditional NI-DAQ has the same VIs and functions and works the same way as NI-DAQ 6.9.*x*. You can use both Traditional NI-DAQ and NI-DAQmx on the same computer, which is not possible with NI-DAQ 6.9.*x*.

U

UDP	User Datagram Protocol. A standard format for transmitting data in datagrams from one computer to another.

V

VI	Virtual Instrument—(1) A combination of hardware and/or software elements, typically used with a PC, that has the functionality of a classic stand-alone instrument; (2) A LabVIEW software module (VI), which consists of a front panel user interface and a block diagram program.
VI Server	Mechanism for controlling VIs and LabVIEW applications programmatically, locally and remotely.
Virtual Instrument Software Architecture (VISA)	Single interface library for controlling GPIB, VXI, RS-232, and other types of instruments.

Index

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W