DAQ

LabVIEW™ RT

PCI/PXI™-7030 and LabVIEW RT User Manual

NATIONAL INSTRUMENTS™

April 1999 Edition
Part Number 322154A-01

Important Information

Warranty

RT Series DAQ hardware is warranted against defects in materials and workmanship for a period of one year from the date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace equipment that proves to be defective during the warranty period. This warranty includes parts and labor.

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

WARNING REGARDING MEDICAL AND CLINICAL USE OF NATIONAL INSTRUMENTS PRODUCTS

National Instruments products are not designed with components and testing intended to ensure a level of reliability suitable for use in treatment and diagnosis of humans. Applications of National Instruments products involving medical or clinical treatment can create a potential for accidental injury caused by product failure, or by errors on the part of the user or application designer. Any use or application of National Instruments products for or involving medical or clinical treatment must be performed by properly trained and qualified medical personnel, and all traditional medical safeguards, equipment, and procedures that are appropriate in the particular situation to prevent serious injury or death should always continue to be used when National Instruments products are being used. National Instruments products are NOT intended to be a substitute for any form of established process, procedure, or equipment used to monitor or safeguard human health and safety in medical or clinical treatment.

Compliance

FCC/DOC Radio Frequency Interference Class A Compliance

This equipment generates and uses radio frequency energy and, if not installed and used in strict accordance with the instructions in this manual, may cause interference to radio and television reception. Classification requirements are the same for the Federal Communications Commission (FCC) and the Canadian Department of Communications (DOC). This equipment has been tested and found to comply with the following two regulatory agencies:

Federal Communications Commission

This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at his own expense.

Notices to User: ***Changes or modifications not expressly approved by National Instruments could void the user's authority to operate the equipment under the FCC Rules.***

This device complies with the FCC rules only if used with shielded interface cables of suitable quality and construction. National Instruments used such cables to test this device and provides them for sale to the user. The use of inferior or nonshielded interface cables could void the user's authority to operate the equipment under the FCC rules.

If necessary, consult National Instruments or an experienced radio/television technician for additional suggestions. The following booklet prepared by the FCC may also be helpful: *Interference to Home Electronic Entertainment Equipment Handbook*. This booklet is available from the U.S. Government Printing Office, Washington, DC 20402.

Canadian Department of Communications

This Class A digital apparatus meets all requirements of the Canadian Interference-Causing Equipment Regulations.

Cet appareil numérique de la classe A respecte toutes les exigences du Règlement sur le matériel brouilleur du Canada.

Contents

Chapter 4 Software Overview

Appendix A LabVIEW RT Function Reference

About This Manual

This manual contains information about the RT Series boards and the LabVIEW RT software. The RT Series hardware is made up of the following boards:

- PCI-7030/6040E
- PCI-7030/6030E
- PCI-7030/6533
- PXI-7030/6040E
- PXI-7030/6030E
- PXI-7030/6533

The RT Series family of boards are multifunction I/O boards with an embedded processor. LabVIEW RT and NI-DAQ create embedded, real-time applications that execute on the RT Series board.

Conventions Used in This Manual

The following conventions are used in this manual:

<>	Angle brackets enclose the name of a key on the keyboard—for example, <shift>. Angle brackets containing numbers separated by an ellipsis represent a range of values associated with a bit or signal name—for example, DBIO<3..0>.
[]	Square brackets enclose optional items—for example, [`response`].
-	A hyphen between two or more key names enclosed in angle brackets denotes that you should simultaneously press the named keys—for example, <Control-Alt-Delete>.
»	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 the beginning of an activity.

■	This icon denotes the end of an activity.
💡	This icon denotes a tip, which alerts you to advisory information.
📝	This icon denotes a note, which alerts you to important information.
⚠	This icon denotes a caution, which advises you of precautions to take to avoid injury, data loss, or a system crash.
⚡	This icon denotes a warning, which advises you of precautions to take to avoid being electrically shocked.
bold	Bold text denotes the names of menus, menu items, parameters, dialog boxes, dialog box buttons or options, icons, windows, Windows 95 tabs, or LEDs.
italic	Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font also denotes text from which you supply the appropriate word or value, as in Windows 3.*x*.
`monospace`	Text in this font denotes text or characters that you should literally 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, and for statements and comments taken from programs.
`monospace italic`	Italic text in this font denotes that you must enter the appropriate words or values in the place of these items.
paths	Paths in this manual are denoted using backslashes (\) to separate drive names, directories, folders, and files.

Related Documentation

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

- *LabVIEW QuickStart Guide*
- *LabVIEW User Manual*
- *LabVIEW G Programming Reference Manual*
- *LabVIEW Data Acquisition Basics Manual*
- *LabVIEW Online Reference*, available by selecting **Help»Online Reference**

Hardware information for the I/O portion of your hardware is found in the hardware user manual for the specific I/O board attached to your RT Series board. For example, if you are using the PXI-7030/6040E, refer to the *PXI E Series User Manual* for information on the 6040E daughter board.

Introduction

This chapter describes the RT Series boards, lists what you need to get started, describes the software you need to use the boards, and explains how to unpack your RT Series board.

About the RT Series

The RT Series boards are multifunction DAQ boards with an embedded processor. The RT Series board, along with LabVIEW RT and NI-DAQ, provide an easy to use system for real-time applications.

Each RT Series board is made up of two parts—a processor board and an DAQ daughter board. The processor board contains an embedded microprocessor that runs real-time, embedded LabVIEW RT applications. The DAQ board provides the same high-performance DAQ as National Instruments non-RT DAQ boards. For example, the PXI-7030/6040E contains an DAQ board that is equivalent to the PXI-6040E.

This manual does not contain detailed information about the DAQ component of the RT Series boards. For this information, you should refer to the appropriate DAQ hardware user manual. Table 1-1 lists the appropriate DAQ board manual for each of the RT Series boards.

Table 1-1. RT Series Board and Corresponding DAQ Manual

RT Series Board	DAQ Manual
PXI-7030/6040E	*PXI E Series User Manual*
PXI-7030/6030E	*PXI E Series User Manual*
PXI-7030/6533	*DIO 6533 User Manual*
PCI-7030/6040E	*PCI E Series User Manual*
PCI-7030/6030E	*PCI E Series User Manual*
PCI-7030/6533	*DIO 6533 User Manual*

What You Need to Get Started

To set up and use your RT Series board, you will need the following:

- One of the following boards:
  - PXI-7030/XXXX
  - PCI-7030/XXXX
- *PCI/PXI 7030 and LabVIEW RT User Manual*
- Hardware user manual for DAQ daughter board
- LabVIEW RT
- NI-DAQ 6.5.2 or greater

Note Be sure to install the NI-DAQ software before you install the hardware.

Unpacking

Your RT Series board is shipped in an antistatic package to prevent electrostatic damage to the board. Electrostatic discharge can damage several components on the board. To avoid such damage in handling the board, take the following precautions:

- Ground yourself via a grounding strap or by holding a grounded object.
- Touch the antistatic package to a metal part of your computer chassis before removing the board from the package.
- Remove the board from the package and inspect the board for loose components or any other sign of damage. Notify National Instruments if the board appears damaged in any way. Do not install a damaged board into your computer.
- Never touch the exposed pins of connectors.

Software Programming Choices

The RT Series boards can be used with LabVIEW RT and NI-DAQ 6.5.2 or greater. NI-DAQ is included with LabVIEW RT and may be installed during the LabVIEW RT installation. Refer to Chapter 2, *Installation and Configuration*, for detailed instructions on installation of your RT Series hardware and software.

Optional Equipment

Refer to the appropriate DAQ board manual (Table 1-1) for optional equipment information.

Custom Cabling

Refer to the appropriate DAQ board manual (Table 1-1) for custom cabling information.

2

Installation and Configuration

This chapter explains how to install and configure your RT Series hardware and software.

Software Installation

Complete the following steps to install LabVIEW RT.

Caution LabVIEW RT is available in English only. If you upgrade from a French, German, or Japanese version of LabVIEW to LabVIEW RT, your version of LabVIEW will be in English.

1. If you are installing LabVIEW RT on Windows NT, log on to Windows NT as an administrator, or as a user with administrator privileges.
2. Insert the LabVIEW RT CD into your CD drive.
3. Run the LabVIEW RT installation program. For users with the LabVIEW RT Full Development System configuration, the installation program is found at `X:\LabVIEW RT\install\disk1\setup.exe`, where *X* is the letter of your CD drive. For users with the LabVIEW RT Professional Development System configuration, the installation program is found at `X:\LabVIEW RT\install\setup.exe`, where *X* is the letter of your CD drive.
4. Follow the instruction that appears on your screen.

Note The LabVIEW RT installation procedure updates some LabVIEW 5.*x* files. Therefore, if you re-install LabVIEW 5.*x*, you must re-install LabVIEW RT.

5. When prompted during the LabVIEW installation, select to install NI-DAQ 6.5.2.
6. Follow the installation instructions that appear on your screen.

Hardware Installation

You can install the RT Series board in any available expansion slot in your computer or PXI chassis. However, to achieve best noise performance, leave as much room as possible between the RT Series board and other boards and hardware. The following are general installation instructions, so consult your computer user manual or technical reference manual for specific instructions and warnings.

PCI Installation

1. Write down the RT Series board serial number for future reference.
2. Turn off and unplug your computer.
3. Remove the cover to your computer.
4. Remove the expansion slot cover on the back panel of the computer.
5. Insert the RT Series board into a 5 V PCI slot. Gently rock the board to ease it into place. It may be a tight fit, but do not force the board into place.
6. Screw the mounting bracket of the RT Series board to the back panel rail of the computer.
7. Replace the cover.
8. Plug in and turn on your computer.

Your RT Series PCI board is installed. You are now ready to configure your software. Refer to Chapter 4, *Software Overview* for instructions on how to configure the LabVIEW RT software.

After the installation program has completed, run the Measurement & Automation Explorer, found in the `NI-DAQ for Windows` program folder. The Measurement & Automation Explorer finds the RT Series board and any other DAQ board you have in your system and assigns device numbers to them. Note that the RT Series board is assigned two device numbers—one for the processor board, and one for the DAQ daughter board. For more information about your RT Series hardware, see Chapter 3, *Hardware Overview*.

Note Remember the device numbers of your RT Series hardware. You need them to download and run LabVIEW RT VIs to your RT Series board.

Save the configuration information by selecting **File»Save**. Then exit the Measurement & Automation Explorer.

PXI Installation

1. Write down the PXI E Series board serial number for future reference.
2. Turn off and unplug your computer.
3. Choose two adjacent unused PXI slots in your system.
4. Remove the filler panels for the slots you have chosen.
5. Insert the RT Series board into the 5 V PXI slots. Use the injector/ejector handle to fully insert the board into the chassis.
6. Screw the front panel of the RT Series board to the front panel mounting rail of the system.
7. Plug in and turn on your computer.

Your RT Series PXI board is installed. You are now ready to configure your software. Refer to Chapter 4, *Software Overview* for instructions on how to configure the LabVIEW RT software.

After the installation program has completed, run the Measurement & Automation Explorer, found in the `NI-DAQ for Windows` program folder. The Measurement & Automation Explorer finds the RT Series board and any other DAQ board you have in your system and assigns device numbers to them. Note that the RT Series board is assigned two device numbers—one for the processor board, and one for the DAQ daughter card. For more information about your RT Series hardware, see Chapter 3, *Hardware Overview*.

Note Remember the device numbers of your RT Series hardware. You need them to download and run LabVIEW RT VIs to your RT Series board.

Save the configuration information by selecting **File»Save**. Then exit the Measurement & Automation Explorer.

Board Configuration

Due to the National Instruments standard architecture for data acquisition and the PCI and PXI bus specification, the RT Series boards are completely software configurable.

The PCI RT Series boards are fully compatible with the industry standard PCI Local Bus Specification Revision 2.0. The PXI RT Series boards are fully compatible with the PXI Specification Revision 1.0. This allows the PCI or PXI system to automatically perform all bus-related configurations and requires no user interaction.

Data-acquisition-related configuration includes such settings as analog input polarity and range, analog input mode, and others. You can modify these settings through the Measurement & Automation Explorer, as well as through NI-DAQ and LabVIEW RT.

3 Hardware Overview

This chapter describes the PCI and PXI RT Series hardware. You will find most of the hardware information you need in the DAQ board manual. Table 1-1 lists the DAQ board manuals for the various RT Series boards.

As shown below, a DAQ daughter board attaches to the processor board, and together they form the RT Series Hardware. The following sections describe the components that make up the RT Series Hardware.

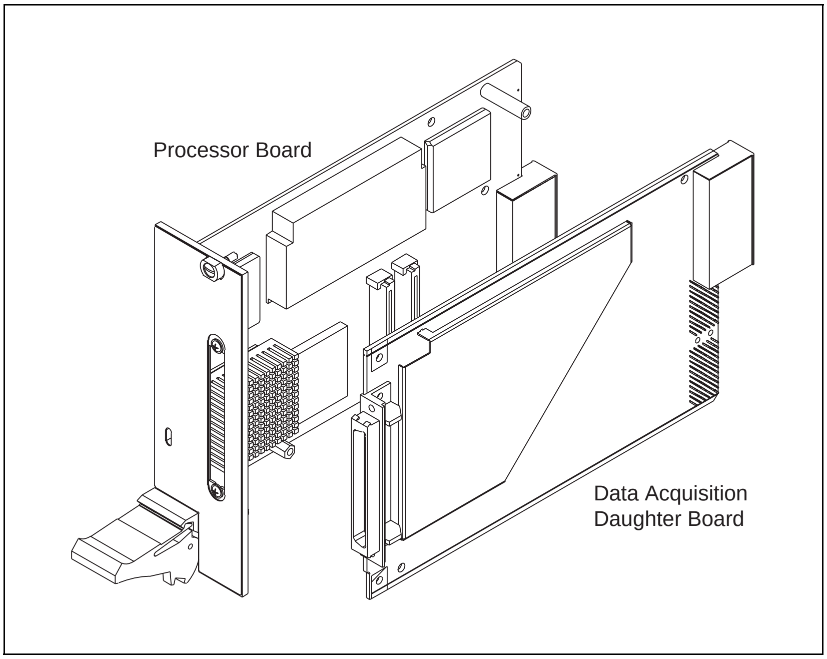


Figure 3-1. PXI RT Series Real-Time Data Acquisition Hardware

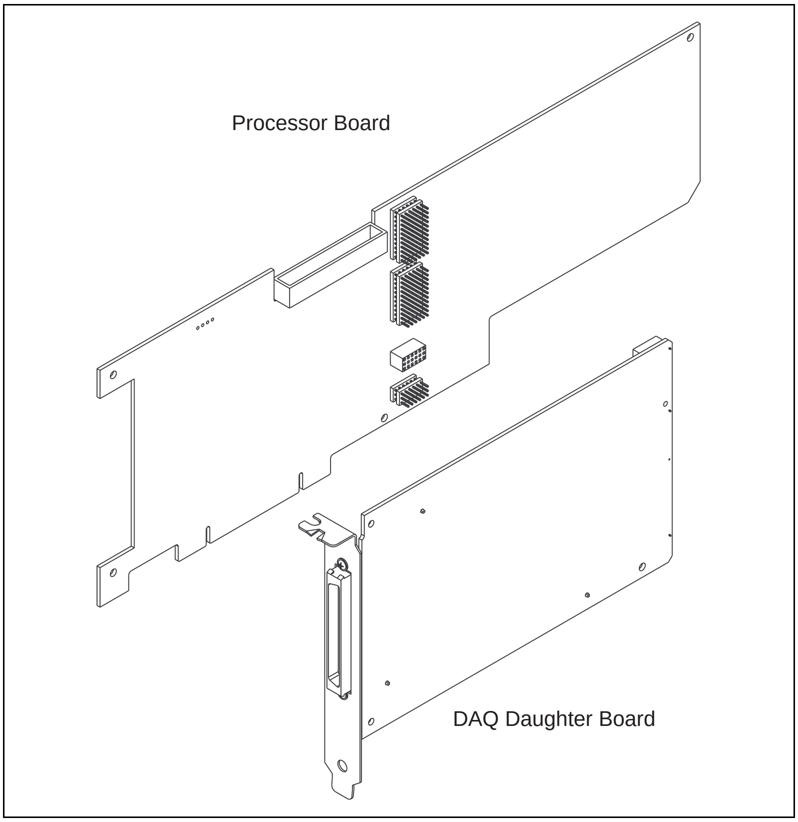


Figure 3-2. PCI RT Series Real-Time Data Acquisition Hardware

Processor Board

The 7030 processor board contains a microprocessor and support circuitry, as shown in Figure 3-3. The support circuitry includes a chipset with PCI controller and ISA bridge, main memory (DRAM), L2 cache (SRAM), and BIOS. The LabVIEW RT software uses this embedded processor system for its execution platform.

The DAQ daughter board resides on the embedded PCI bus. This provides a high-performance system for communication between the software running on the processor and the DAQ daughter board.

The processor board also holds the shared memory that interfaces between the host system and the embedded processor. Refer to Chapter 4, *Software Overview*, for more information about using shared memory for your applications.

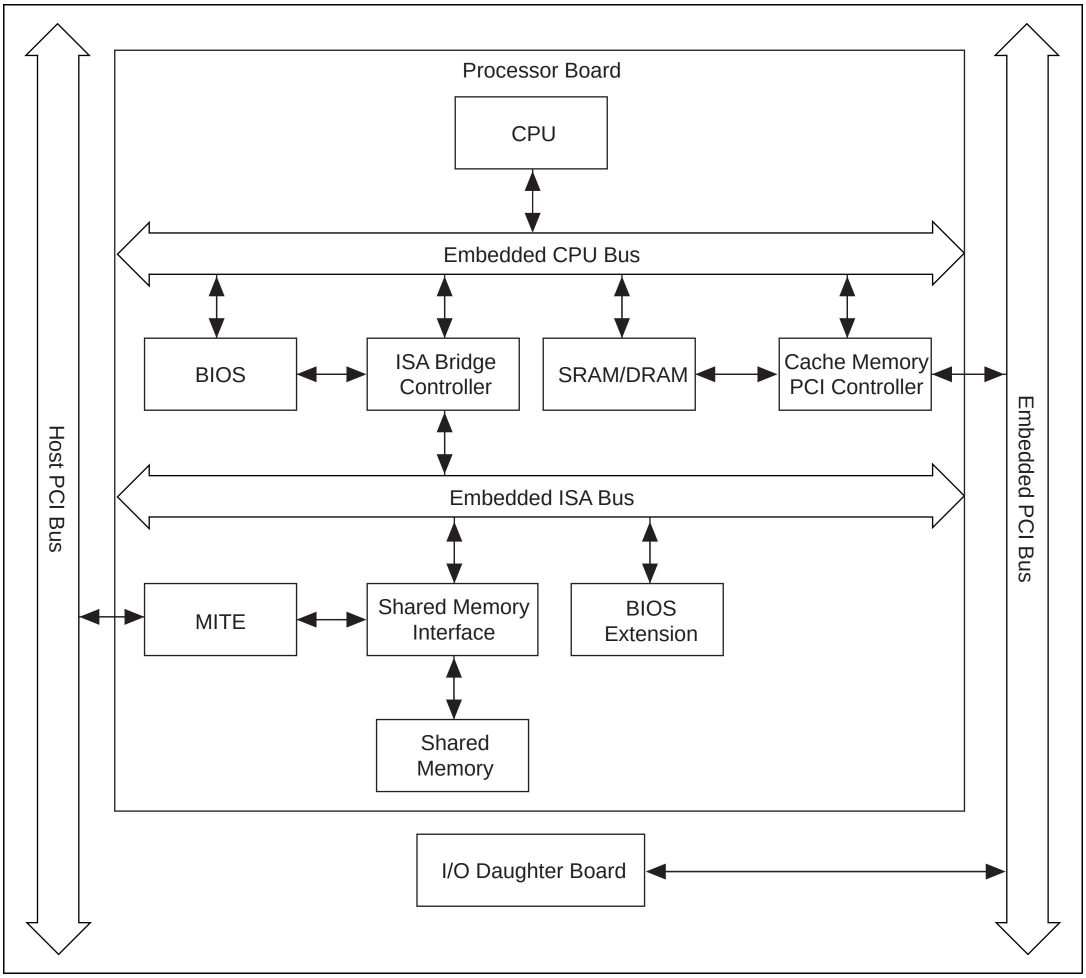


Figure 3-3. RT Series Data Acquisition Board Block Diagram

DAQ Board

The daughter board in the RT Series system provides the DAQ functionality. Each daughter board is a National Instruments PXI board with some mechanical modifications to allow it to connect to the processor board. This means that you can expect the same high performance and specifications from the daughter board that you would from the standard PXI version of the board. For example, the 7030/6040E has the same DAQ characteristics as the PXI-6040E. For this reason, you should refer to appropriate daughter board manual for specifications, cabling requirements, and accessory information. Table 1-1 lists the appropriate manual for each available daughter board. Additional information on hardware and software configuration is available in the DAQ section of the *LabVIEW Online Reference*, available by selecting **Help»Online Reference…**.

Host to RT Series Board Communication

The RT Series board communicates to the computer in which it resides through shared memory on the 7030 processor board. The computer in which you use the board is called the host. Both the host computer and the embedded processor have access to the shared memory. The memory is allocated for various tasks. For example, LabVIEW RT uses part of the shared memory for downloading software to the board. There is also some memory space allocated to the user for passing data between the host PC and the processor board. Refer to Chapter 4, *Software Overview*, for more information on passing parameters and shared memory.

RTSI

The real-time system integration bus (RTSI™ bus) directly connects DAQ boards for precise synchronization of functions. All RTSI functionality is available on the RT Series boards. The RTSI bus allows you to connect RT Series boards to any other DAQ board regardless of whether the other board is an RT board or standard DAQ board.

The RTSI connections are made across the PXI trigger bus for the PXI RT boards. For PCI RT boards, the connection is made with a connector on the top edge of the board and a RTSI cable available from National Instruments. Refer to your daughter board user manual for information on how to use the RTSI signals. Table 1-1 lists the appropriate manual for each available daughter board.

LEDs

There are two LEDs on the RT processor board. These are user-controlled LEDs that you can use to indicate the state of your running application. On the PXI-7030, the LEDs are visible on the front panel. On the PCI-7030, the LEDs are located along the top edge of the board, and you must remove the cover from the case of your computer to view them.

The red LED is turned on during the reset of the the RT Series board, and turns off when the reset is finished. Also, the red LED turns on and remains lit if the RT Engine detects an internal error.

You can use the RT Board LEDs VI, located in the **Functions»RT»Shared Memory Utilities** palette, in your application to control each LED independently. You may turn each LED on, off, or toggle its state. For more information on the RT Board LEDs VI, see Appendix A, *LabVIEW RT Function Reference*.

4 Software Overview

This chapter describes the software you use to create LabVIEW RT applications and provides an overview of real-time programming.

Introduction

Most LabVIEW applications run on general-purpose operating systems like Windows NT/98/95. Some LabVIEW applications require deterministic real-time performance that Windows 95/98/NT cannot guarantee. National Instruments created LabVIEW RT to address the need for deterministic real-time performance in LabVIEW.

LabVIEW RT combines the ease-of-use of LabVIEW with the power of real-time systems, so you can generate deterministic applications using graphical programming. These real-time applications are executed on RT Series boards, which combine data acquisition with the processing power of a computer.

General-purpose operating systems can “crash” or “hang,” which causes programs to quit running. Because LabVIEW RT applications run on a separate hardware platform, they do not stop executing if the PC’s operating system crashes. If a crash occurs on the host PC’s operating system, the user interface is lost, and any other communication between an embedded LabVIEW RT application and the host PC cease. However, embedded LabVIEW RT applications continue to run, even after the operating system crashes.

You can *soft reboot* the host PC without disrupting embedded LabVIEW RT applications. After you reboot the host PC, you can reestablish the host PC’s communication with LabVIEW RT. You can also design your applications so you can retrieve any data that was collected on the RT Series board while the host PC was not in communication with the RT Series board.

Architecture of LabVIEW RT

The following figure shows the components of LabVIEW RT. In this architecture, LabVIEW RT applications run on RT Series hardware.

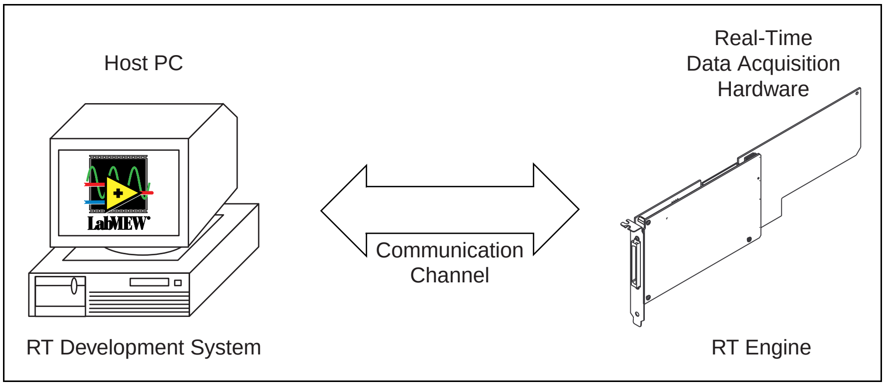


Figure 4-1. Components of LabVIEW RT

LabVIEW RT consists of two components: the *RT Development System* and the *RT Engine*, which communicate with each other via the built-in *communication channel*. The RT Development System initializes the board, downloads VIs to the RT Series hardware, and provides a user interface for debugging LabVIEW RT applications. The RT Engine runs on the RT Series hardware and executes the LabVIEW RT VIs in real-time.

RT Development System

The RT Development System is a Windows NT/98/95 application that runs on the host computer. When you boot the RT Series hardware, it has no VIs to run. You can use the RT Development System to download VIs to the RT Series board.

After LabVIEW RT VIs are downloaded and running, you can close the RT Development System—the RT Engine keeps the embedded VIs running. However, you can keep the RT Development System open to show the front panel of the embedded LabVIEW RT VI, providing you with a user interface. When VIs run in the RT Series hardware, The RT Development System exchanges messages with the RT Engine to update the controls and indicators on the front panel. These communications are built into the RT Development System and embedded software, so they occur automatically.

You can use the RT Development System for debugging embedded LabVIEW RT VIs, even while the embedded LabVIEW RT VIs run in the RT Series hardware. All the normal LabVIEW debugging facilities, such as probes, breakpoints, and single stepping, are supported.

RT Engine

The RT Engine runs the LabVIEW RT VIs you download with the RT Development System and provides real-time performance. LabVIEW RT can provide deterministic real-time performance for the following reasons:

- The RT Engine runs on a real-time operating system, which ensures that the scheduler and other operating system services adhere to real-time operation.
- The RT Engine does not run on the host computer, but on the RT Series hardware. Furthermore, other than LabVIEW RT and the NI-DAQ driver, no other application or device drivers (such as disk or network drivers) run on the platform. Not having extraneous software and drivers means LabVIEW RT is not impeded by a third-party application or driver.
- The RT Engine is tuned for real-time performance, and bottlenecks are reduced or eliminated.
- The RT Series hardware uses no virtual memory, thereby removing a major source of unpredictability in deterministic systems.

System Time/Local Time on the RT Engine

Time and date information on the RT board is set whenever you start or reset the board. However, due to limitations of the real-time operating system on the RT Series hardware, the RT Engine does not support time zone information. As a result, system time and local time are treated as being the same in the RT Engine. Therefore, the local time of the host is both the local time and system time of RT Engine.

As a result, when a chart’s axis marker is formatted to show Time and Date, it displays values equal to Greenwich Mean Time (GMT).

Communication Channel

Part of the onboard memory (RAM) on the RT Series board is accessible to both the host PC and the RT Series board. This part, called the *shared memory*, is the medium through which the host computer communicates with the RT Series board. For more information on shared memory communications, refer to the *Shared Memory* section later in this chapter.

Operating LabVIEW RT

This section describes basic operating procedures for creating LabVIEW RT applications.

Launching LabVIEW RT and Downloading VIs

When you launch LabVIEW RT, the following dialog box appears.

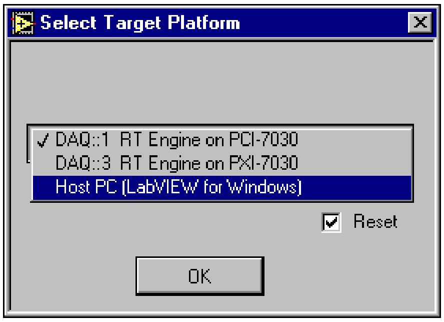

Select Target Platform
✓ DAQ::1 RT Engine on PCI-7030
DAQ::3 RT Engine on PXI-7030
Host PC (LabVIEW for Windows)
Reset
OK

Figure 4-2. Select Target Platform Dialog Box

Use the pull-down menu to select where you want to execute VIs. When you select an RT Series board, any VI you subsequently run is downloaded to that RT Series board, where it can be executed. You can also select **Host PC (LabVIEW for Windows)**, which starts LabVIEW on the Host PC without the embedded LabVIEW RT capabilities.

Select the **Reset** checkbox to reset the RT Series board. Any VIs already on the selected RT Series board are stopped and removed from the RT Engine when you reset.

When you select an RT Series board and click **OK**, the RT Development System establishes a connection with the RT Series board you selected. From this point on, you can open VIs and hit the **Run** button—the RT Development System automatically downloads the VI and its associated subVIs to the RT Series hardware. The RT Engine then executes the VI.

Resetting the RT Series Board

Resetting the RT Series board is necessary only after the system is first powered on or if communication with the RT Engine cannot be established or has been lost, such as when you abort an embedded LabVIEW RT VI. You can also reset the board when you change the DAQ configuration information for the RT Series board.

Downloading VIs without Running Them

You can download LabVIEW RT VIs without executing them by selecting **Operate»Download Application**. Do this if you want to use the VI Server mechanism to execute these VIs later.

To see which VIs have been downloaded to your RT Series hardware, select **Project»Show VI Hierarchy** from the RT Development System. In this view, the VI hierarchy appears, with a push pin in the upper left corner of each VI. When the thumb pin is in the vertical position, as shown on the left, the VI has been downloaded.

When the thumb pin is in the horizontal position, the VI needs to be downloaded in order to run.

The VI Server can execute VIs on a remote platform—in this case, the RT Series hardware—but the VI Server cannot download VIs. Therefore, those VIs must be downloaded beforehand. For more information on the VI Server, see the *Distributed Computing and VI Servers* section later in this chapter, and Chapter 21, *VI Server*, of the *G Programming Reference Manual*.

Debugging LabVIEW RT VIs

You debug LabVIEW RT VIs the same way you debug any VI. See Chapter 5, *Debugging*, in the *LabVIEW QuickStart Guide*, and the *How Do You Debug a VI* section in Chapter 2, *Creating VIs*, of the *LabVIEW User Manual*, for information on the debugging features found in LabVIEW. All the existing debugging tools, except for the Call List window, are available with LabVIEW RT.

Building Stand-Alone Executables Using LabVIEW RT and the Application Builder

You cannot build executables when you are operating LabVIEW RT on the RT Series board. The only way you can build application is to launch LabVIEW RT on the host PC from the Select Target Platform dialog box.

Applications built using LabVIEW RT Development system can download VIs to the board. When you launch the built executable, the Select Target Platform dialog box appears. You can select any RT Series board in your system to run your built application.

To disable the Select Target Platform dialog box, specify the board number in the command line argument of your built executable, for example:

```
c:\mybuiltApp.exe -rte DAQ::3
```

You can also reset the board you select by entering `-rtereset`:

```
c:\mybuiltApp.exe -rtereset DAQ::3
```

Programming LabVIEW RT

A LabVIEW RT application runs on the RT Series board and typically controls or monitors external events through its I/O (such as analog and digital I/O channels of a DAQ board).

Because it runs on a hardware platform which does not have all the components found in a PC, LabVIEW RT lacks some features found in LabVIEW. For example, there is no disk drive on the RT Series board, so file I/O is not supported. The following LabVIEW functions are not currently supported in LabVIEW RT:

- ActiveX
- Data logging
- Dialog boxes
- Ethernet
- File I/O
- Instrument I/O: VISA, 488, Serial
- Printing
- Profiler
- Programmatic Menu

Note If you attempt to download and execute to your RT Series board a VI that has any of the unsupported functionality listed above, the VI still executes. However, unsupported functions do not work and return standard LabVIEW error codes.

There are two ways to provide a user interface to LabVIEW RT VIs: using the RT Development System or writing host LabVIEW applications.

Using the RT Development System

Using the RT Development System is the most straightforward way to provide a user interface for LabVIEW RT applications. The RT Development System comes with your LabVIEW RT software package, so you do not have to write a new application. The RT Development System can show the controls and indicators of the embedded LabVIEW RT VIs running on the RT Series board. The user can interact with the VIs by changing the value of the controls, and he or she can see the updates to the indicators.

The RT Development System is most useful during program development and debugging, but there are reasons it may not be as useful for deploying embedded real-time applications.

First, exchanging messages between the RT Development System and the RT Engine runs at a lower priority than a real-time algorithm. Therefore, if a high-priority thread, or "task," uses up all the RT Series board's system resources, no resources are available for lower-priority tasks, which can cause the user interface to appear "locked up." For more information on priorities, see the *Real-Time Programming* section later in this chapter.

Second, in many applications you might want to do more than just display embedded LabVIEW RT data. You might want to save data to disk and analyze the data, or control the behavior of embedded LabVIEW RT applications based on the data gathered. These are functions the RT Development System cannot perform. The RT Development System can display the front panel, but it cannot execute any code.

Getting RT Engine Information

To get the hardware device number for the RT Series hardware to which the RT Development System is currently connected, select **Operate»RT Engine Info**. The device number is also displayed in the lower left corner of the front panel.

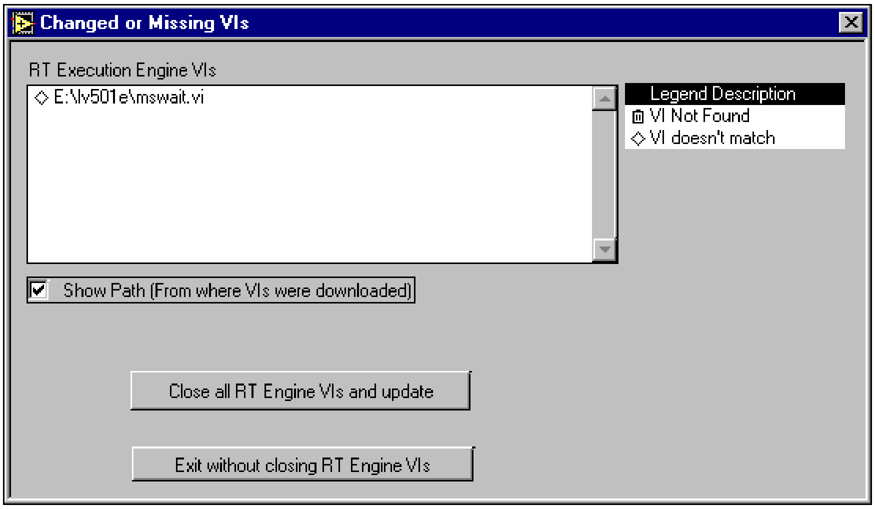
Exiting the RT Development System

You can exit the RT Development System without stopping the embedded LabVIEW RT VIs running in the RT Series hardware by selecting **File»Exit Without Closing RT Engine VIs**. Doing this exits the RT Development System on the host PC, but the VIs running in the RT Series hardware continue running. If a VI is downloaded but not running, the VI remains in the RT Series hardware. Using the VI Server from a host LabVIEW application, you can call and execute the embedded VI later on.

When you quit the RT Development System by selecting **File»Exit**, a confirmation dialog box appears, prompting you to confirm that you want to shut down the RT Engine before exiting. If you select **OK**, the RT Development System closes all the VIs running in the RT Series hardware and shuts down the RT Engine.

Restarting LabVIEW RT with Embedded LabVIEW RT VIs Already Running

If you restart LabVIEW RT and establish a connection to the RT Series hardware while embedded LabVIEW RT VIs are still running, the RT Development System is notified there are VIs running on the RT Series hardware. The RT Development System attempts to open the host PC's copy of the embedded VIs to provide the front panel of the embedded VIs. If the host PC's copies of the VIs have been moved or modified since they were downloaded to the RT Series board, the RT Development System shows the following panel.


Changed or Missing VIs

RT Execution Engine VIs

◇ E:\lv501e\mswait.vi

Legend Description
🗑 VI Not Found
◇ VI doesn't match

[✓] Show Path (From where VIs were downloaded)

Close all RT Engine VIs and update
Exit without closing RT Engine VIs

The dialog box shows the name of the VIs and indicates either that the RT Development System cannot find the VIs, or the VIs have been modified (they do not match the embedded VIs running in the RT Series hardware). From this dialog box, you can either close all the VIs running in the RT Series hardware and update them all with the latest version of each VI, or you can leave the embedded VIs running and quit the RT Development System.

Activity 4-1. RT Development System as User Interface

In this example, the RT Development System shows the front panel of the VI running in the RT Series hardware. You use the front panel as the user interface.

1. Start LabVIEW RT. From the Select Target Platform dialog box, select the appropriate RT Series board. Click **OK**.
2. Open the Tank Simulation VI located in `Examples\Apps\tankmntr\Tank Simulation.vi`. Click **Run.**

   The Tank Simulation VI downloads to the RT Series hardware, which executes the VI. Experiment with the control values to control the Tank Simulation example.
3. Exit LabVIEW RT by selecting **File»Exit Without Closing RT Engine VIs**.

   This exits the RT Development System but leaves the embedded Tank Simulation VI running on the RT Series Hardware.
4. Start LabVIEW RT again. From the Select Target Platform dialog box, select the same RT Series board. Click **OK**.

   Since you chose to exit without closing the embedded VIs, LabVIEW RT reestablishes communication with the RT Series board, opens the Tank Simulation VI in the RT Development System, and provides the RT Development System with the most recently updated data from the embedded LabVIEW RT VI still running on the RT Series board.
5. Exit LabVIEW RT by selecting **File»Exit**.

   This exits LabVIEW RT and stops and closes the Tank Simulation VI on the RT Series hardware.

End of Activity 4-1.

Activity 4-2. Interaction of Embedded VI with Local Copy

1. Start LabVIEW RT and select the appropriate RT Series board from the Select Target Platform dialog box.
2. Select **File»Open** and open `Peek Poke I32 - RT Engine.vi` from `Examples\RT\RTComm.llb.`
3. Select **File»Save As…** and save the VI to your desktop.
4. Download the VI by selecting **Operate»Download Application**.
5. Select **File»Exit Without Closing RT Engine VIs**.
6. Delete the VI you just saved to the desktop.
7. Launch LabVIEW RT and select the appropriate RT Series board from the Select Target Platfrom dialog box. Do NOT select **Reset**. Because the RT Development System cannot find the local copy of the VI, it prompts you to locate it. Select **Cancel**. The Changed or Missing VI dialog box is displayed.

When the RT Development System detects that there is an embedded VI on the RT Series board, it attempts to open the front panel from the local copy of the VI on the host PC hard drive. If the RT Development System detects a change in version, or can not locate the host copy of the VI, the **Changed or Missing VI** dialog box is displayed. You then have the option to close all the embedded VIs and update them with the copy on the host PC, or to exit the RT Development System and not update the embedded VIs.

8. Click on the **Close all RT Engine VIs and Update** button from the Changed or Missing VI dialog box.
9. Close all VIs and exit.

End of Activity 4-2.

Host LabVIEW Applications

Instead of using the RT Development System to communicate with embedded LabVIEW RT VIs, you can write a *host LabVIEW application.* A host LabVIEW application can control the behavior of the embedded LabVIEW RT applications programmatically and save and analyze data produced by an embedded LabVIEW RT VI. A host LabVIEW application also provides a user interface while communicating with the embedded LabVIEW RT application.

You can develop host LabVIEW applications with LabVIEW RT by selecting **Host PC (LabVIEW for Windows)** at the Select Target Platform dialog box. Doing this makes LabVIEW RT behave like regular LabVIEW on the host PC.

Shared memory is the physical medium in which the host computer and RT Series hardware communicate. However, there are alternative high-level software protocols available to host LabVIEW applications for communication with LabVIEW RT VIs. Each protocol has its advantages and drawbacks. You can choose any one of them based on your communication needs.

Shared Memory

In operating systems like Windows NT/98/95, two processes or applications can communicate with each other using the shared memory mechanism the operating system provides. Similarly, embedded LabVIEW RT applications and host LabVIEW applications can communicate using shared memory VIs to read and write to the shared memory locations on the RT Series hardware.

Shared memory VIs have very low overheads and are ideal for time-critical, real-time applications. However, the size of the shared memory is limited to 1 kilobyte, so if you need to transfer several megabytes of data, you must break up the data into smaller chunks and then transfer them. In doing so, you must make sure that data in the shared memory is not overwritten before it is read. The TCP/IP VIs are more convenient for transferring large amounts of data.

TCP/IP

TCP/IP is an industry standard protocol for communication over networks. LabVIEW supports TCP/IP, and LabVIEW RT extends the capability of the existing TCP/IP VIs, so you can use these VIs not only to communicate across the network, but across shared memory as well. Therefore, a host

LabVIEW application can communicate with an embedded LabVIEW RT VI using the TCP/IP VIs.

There are advantages to using TCP/IP VIs for communication. TCP/IP VIs manage flow control, so they are more convenient for bulk transfer—the TCP/IP VIs automatically break-up the transfer into smaller sizes. TCP/IP VIs are also more portable and medium independent, and work over both shared memory and the network. By comparison, shared memory access is limited to communicating over shared memory.

The drawback of TCP/IP is that it has higher overhead than shared memory access and so might not be suitable for some fast real-time applications.

Distributed Computing and VI Servers

VI Server is a LabVIEW mechanism used to execute VIs on a remote machine across a network. Using VI Server technology, a host LabVIEW application views the RT Series hardware as just another machine on which it can invoke VIs. As the host LabVIEW application calls LabVIEW RT VIs through the VI Server, the host LabVIEW application can conveniently pass parameters in and out of the embedded LabVIEW RT VIs. The VI Server takes care of packaging the parameters and shuttling the parameters between the host LabVIEW application and the embedded LabVIEW RT VIs.

There are advantages to using the VI Server for communication. The VI Server allows you to access the functionality of TCP/IP, while working within the framework of LabVIEW. For more information on the VI Server, see Chapter 21, *VI Server,* of the *G Programming Reference Manual*.

Like TCP/IP VIs, the VI Server manages flow control, so it is more convenient for bulk transfer—the VI Server automatically breaks up the transfer into smaller sizes. The VI Server is also more portable and medium independent, and works over both shared memory and the network. By comparison, shared memory access is limited to communicating over shared memory.

The drawback of the VI Server is that it has higher overhead than shared memory access and TCP/IP, and so might not be suitable for some fast real-time applications. Another limitation of the VI Server is that the host LabVIEW application only can call LabVIEW RT VIs that are already loaded into the RT Series hardware. The host LabVIEW application cannot dynamically download LabVIEW RT VIs to the RT Series hardware. Currently, the only way to download VIs to the RT Series hardware is through the RT Development System.

Activity 4-3. Host LabVIEW Application as User Interface

The TCP/IP Client/Server example shows you how to use a separate host application as the user interface. In this example, there are two VIs—the Client VI and the Server VI. The Server VI runs on the RT Series board and communicates with the Client VI that runs on the host computer via a host LabVIEW application. The Client VI provides the user interface.

1. Start LabVIEW RT. From the Select Target Platform dialog box, select the appropriate RT Series board. Click **OK**.
2. Open the Simple Data Server VI, found in `Examples\Comm\tcpex\Simple Data Server.vi` and click **Run**.

   The Simple Data Server VI downloads automatically to the RT Series hardware, which executes the VI.
3. Select **File»Exit Without Closing RT Engine VIs**. This exits the RT Development System but leaves the embedded TCP/IP Server VI running on the RT Series Hardware.
4. Start LabVIEW RT. In the Select Target Platform dialog box, select **Host PC (LabVIEW for Windows)** to use the normal LabVIEW to execute VIs, rather than the RT Engine on the RT Series board.
5. Open the Simple Data Client VI, found in `Examples\Comm\tcpex\simple data client.vi`.
6. On the Simple Data Client VI front panel, change the Address control from `localhost` to `DAQ::`*`x`*, where *`x`* is the device number of the desired RT Series processor board on your system.

Tip When you access shared memory, the device number you use depends upon what type of operation you are attempting. When communicating from the host PC to an embedded VI, use the device number for the processor board (7030). When performing DAQ operations, use the device number for the DAQ daughter board. When accessing shared memory from an embedded VI on the RT Engine, use device number = 0.

7. Click **Run**.

Because you selected the Host PC rather than the RT Engine in the Select Target platform dialog box, the TCP/IP Client VI runs on the host computer. It receives and displays data through TCP/IP from the Simple Data Server VI running on the RT Series board.

8. Select **File»Exit** to exit LabVIEW RT.

End of Activity 4-3.

Real-Time Programming

This section describes how to program real-time virtual instruments in LabVIEW RT using the scheduling priority of a VI and the methods of communication between the RT Series hardware and host LabVIEW VIs. This section also outlines tips you can use to get the best possible real-time performance out of the RT Series hardware to write high performance deterministic control programs needed in time-critical control loops.

Real-Time Performance of VIs

LabVIEW RT executes VIs in a deterministic manner. In other words, VIs execute with the same time characteristics each occurrence of the VI. For example, control loops execute at a consistent frequency every iteration of the loop. In order to achieve optimal real-time performance, you will need to understand certain programming concepts.

Time Critical Priority

In a real-time operating system, events and interrupts are prioritized, with higher priority events executed before lower priority events.

In a real-time environment, such as running on the RT Series hardware, higher priority tasks always get executed before tasks with lower priorities. Therefore, to ensure your VI executes in a time-critical fashion, set the priority of the VI to time-critical. To set a VIs priority, complete the following steps.

1. Right-click on the VI icon and select **VI Setup…**.
2. Select **Execution Options** from the pull-down menu.
3. Select **time critical priority (highest)**.

LabVIEW RT Real-Time Environment

Since running a VI at higher priority than the RT environment means a VI gets more processor time and resources than the RT environment, setting a VI to run at time-critical priority can have unexpected consequences. Although the RT environment is a multithreaded environment, running a VI at time-critical priority can cause it to monopolize the RT Series processor, making other threads unable to run.

Both the front panel updates in the RT Development System and VI Server or TCP/IP communication run as separate threads on the RT Series hardware. Therefore, a VI running at time-critical priority may prevent these threads from running. Other VIs written by the user running on the RT series hardware may also be affected.

The metronome and stopwatch, found in the **Function»Time and Dialog** palette, cause a VI's execution thread to "sleep," or pause, for a period of time so other threads such as the communication thread have a chance to run. In general, the metronome is preferred over the stopwatch, since it can be used to set a known loop time. For example, placing a metronome in a while loop inside a VI with a constant of 5 wired to it makes the VI try to execute the loop once every 5 ms, or with a loop rate of 200 Hz. If the VI is running at time-critical priority, such a loop rate is guaranteed, assuming that the loop can complete in that given amount of time. Any extra time left over is then given back to the processor to run other tasks. If there is no extra time left over, no other threads will run. In this way, even a VI which waits may be too complicated to yield and actually let other threads run.

Note No method currently exists to allow a VI to wait for less than 1 ms. As such, no VI with a wait inside a loop can execute that loop faster than 1 kHz.

Running a VI at Time-Critical Priority in the RT Development System

If you run at time-critical priority a VI which doesn't wait, it monopolizes the processor on the RT Series hardware. This can make you unable to interact with the VI's front panel and unable to stop the VI using the abort button. There is no problem with the VI or LabVIEW RT if this occurs, it merely means that the RT Series board is too busy to communicate with the RT Development System while the time-critical VI is running. To avoid this happening during development and testing of your VI, set the VI to a lower priority than is actually desired. After development is completed, you can set the priority to the appropriate level.

Running a VI which waits at time-critical priority is often acceptable during development, as the RT Development System can use the time the thread is sleeping to perform communication. However, if the VI is computation-intensive or is unable to complete in the time allotted to it, then no wait occurs, and the RT Development System is unable to communicate with the RT Series board. Even if the loop is able to complete, but leaves only a small fraction of the time available on the processor to the RT Development System communication thread, interaction with that VI will seem very sluggish or nonresponsive.

If your embedded LabVIEW RT VI uses all the processor resources on the RT Series board, you will not be able to communicate to the embedded VI.

The RT Development System then displays a dialog box informing you communication between the RT Development System and the RT Engine has been lost. From this dialog box, click the **Abort** button to close the RT Development System so you can reset the RT Series board on restart.

Running a VI at Time-Critical Priority without the RT Development System

One way to improve performance is to run the VI on the RT Series hardware without the RT Development System. When in this mode, the VI only needs to yield as much time as is required for its TCP/IP or VI Server communication, which can be substantially less time than required for the full RT Development System to communicate. A VI could also *peek and poke* to shared memory, and thus not have to yield any processor time, which gives the maximum possible performance out of the RT Series hardware. Peeking, or reading directly from shared memory, and poking, or writing directly to shared memory, is a common programming technique that can be especially useful for developing LabVIEW RT applications. Deterministic control loops in excess of 1kHz are possible when this approach is used.

Note While Windows NT does not support peek and poke to the shared memory on the host PC, the shared memory on the RT Series processor board does support peek and poke operations.

Activity 4-4. Learning Priorities

In a real-time operating system like the one used to power the embedded RT Engine, events are prioritized and higher priority events are executed over lower priority events. Therefore, VI priorities and threads become very important in developing a real-time application. Failure to use good priority settings could result in VIs which use all the processors resources and starve other lower priority VIs. These low priority tasks and threads, like communication, or the user interface thread, may not even be allowed to run.

1. Launch LabVIEW RT and select the appropriate RT Series board from the Select Target Platform dialog box, and make sure **Reset** is also checked to reboot the RT Series hardware and click **OK**.
2. Select **File»Open** and open the `Priority Trouble.vi` from `Examples\RT\RTTutorial.llb`.

3. In the **VI Setup…** under Execution Options, notice that the priority is set to time critical priority (highest). This makes the diagram of the VI run in the highest priority thread on the RT Engine.
4. On the front panel, make sure the **msec to wait** control is set greater than 0 (10 is the default). Run the VI. The chart displays a sine wave.
5. Slowly decrement the **msec to wait**. With each step, the chart display should speed up, as expected. However, when the **msec to wait** count reaches zero, the front panel will no longer update.

The reason for this behavior is that the RT Series board can no longer communicate with the RT Development System front panel. Since the time critical loop has the highest priority, the user interface thread that updates front panel objects cannot get any processor time. As a result, the front panel and VI appears to be locked—however, the diagram is still running.

6. Click on the abort button. After a short period of time, the RT Development System recognizes that it has lost communication with the processor card, and displays a warning message. Click **OK** to exit LabVIEW RT.

End of Activity 4-4.

Configuration Issues

The RT Engine is downloaded to the RT Series hardware when the hardware is reset, so configuration information is sent at that time. If you make any change to the board's configuration information from the Measurement and Automation Explorer, you must restart LabVIEW RT and reset the RT Series hardware for any changes to be effective.

Virtual Channels, SCXI, and Accessories are also supported on the RT Series hardware, but their descriptions are downloaded to the board when the board is reset. If you add a new channel or change a channel's parameters using the Measurement and Automation Explorer, you must reset the RT Series hardware in order for these changes to take effect. However, unlike Virtual Channels, board parameter or accessory configuration information can be set programmatically inside a VI without the need to run the Measurement and Automation Explorer. Therefore, you can programmatically change these parameters in your VI, without the need to reset the hardware.

For more information on configuration, refer to your NI-DAQ documentation and the DAQ section of the *LabVIEW Online Reference*, available by selecting **Help»Online Reference…**.

Performance of LabVIEW RT Series VIs

There are a number of programming paradigms you can use to greatly increase the performance of any LabVIEW program, but these take on special importance with LabVIEW RT, since performance and deterministic real-time requirements often go hand in hand. In addition, since priorities and communication play a major role in LabVIEW RT, understanding them is crucial to any high performance programming.

Additional information about performance can be found in Chapter 28, *Performance Issues*, of the *G Programming Reference Manual*.

Use of Shared Memory

Using peek and poke to write directly to shared memory offers the best performance. Since shared memory can be manipulated both from a host LabVIEW application and an embedded LabVIEW RT VI, a control loop style program can be written which doesn’t have to slow down or yield in order to communicate with the host system. Many examples of such control loops are in the `Examples\RT\RTControl.llb` directory.

Writing Good Loops

The most time-intensive part of any DAQ operation on the RT Series hardware is the configuration. If possible, try to use the intermediate or advanced DAQ VIs in order to move configuration outside of your main loop. If you do use the Basic DAQ VIs, wire the loop iteration number to the VI so configuration only happens once. In addition, if a control or indicator is placed within a loop, and that VI is running in the RT Development System, the VI tries to communicate between the RT Series hardware and the LabVIEW RT Development System running on the main computer. This does not affect performance if the VI is running at time critical priority, but it does hurt the performance of that communication, making the Development System seem sluggish or not-responsive as mentioned above.

Keep redundant or unnecessary computations out of the loop where you need performance. For example, LabVIEW RT provides a set of VIs which use peek and poke to pass an array of data between the host application and the RT Series hardware by passing only a single element with each iteration. This “smearing” of the cost to transfer an array often improves performance dramatically, since only one read or write is necessary per loop iteration, rather than several.

One of the priority options available for subVIs is **subroutine**. A subVI at subroutine priority executes in the thread of its parent VI. This saves on the overhead of calling the VI, making things run smoother and faster. After your subVI is tested, set its priority to **subroutine**, but remember your highest level VI cannot have subroutine priority; set its priority to time-critical.

Avoid array copying, as this requires a linear, rather than a constant, amount of time. Ways of avoiding array copying are passing in initialized arrays of the expected size to a control loop rather than creating arrays using the build array several times over.

Test and experiment several times to determine exactly how long each iteration of the loop is taking, and if possible, try to yield the maximum amount of time in each wait to enable the communication with the RT Series hardware to be as responsive as possible.

Tip Use the Benchmark Shell VI, located in the `Examples\RT\RTTutorial.llb` directory, to accurately measure the time it takes your VI to execute.

For more information on performance issues using LabVIEW, see Chapter 28, *Performance Issues*, in the *G Programming Reference Manual*.

Activity 4-5. Communicating to an Embedded VI

Embedded LabVIEW RT applications and host LabVIEW applications can communicate using shared memory VIs to read and write to the shared memory locations. Shared memory VIs have very low overheads and are ideal for time-critical, real-time applications.

1. Open LabVIEW RT and select the appropriate RT Series board from the Select Target Platform dialog box. Run the `Peek Poke I32 - RT Engine.vi`, found in the `Examples\Rt\RTComm.llb` directory.
2. Select **File»Exit Without Closing RT Engine VIs** so as not to close the embedded VI.

3. Open LabVIEW RT and select **Host PC (LabVIEW for Windows)** from the Select Target Platform dialog box. Run the `Peek Poke I32 - Host PC.vi`, found in the `Examples\RT\RTComm.llb` directory.

   Read the **Show VI Info…** of these VIs for more information.

Tip While peek and poke are low level, and the fastest form of communication, using several of them in a control loop can slow down your application. If you need a loop with communication to run at maximum rates, use the `RT incremental single write.vi` and `RT incremental single read.vi`, found on the **RT»RT Shared Memory Utilities** palette.

The size of the peek/poke shared memory is limited to 1 kilobyte, so if you need to transfer several megabytes of data, you must break up the data into smaller chunks and then transfer them. In doing so, you must make sure that data in the shared memory is not overwritten before it is read. TCP/IP VIs manage flow control, so they are more convenient for bulk transfers.

4. Open and run the `TCP - RT Engine.vi`, found in the `Examples\RT\RTComm.llb` directory, on the RT Engine of your board.
5. Select **File»Exit Without Closing RT Engine VIs** so as not to close the embedded VI.
6. Open and run the `TCP - Host PC.vi`, found in the `Examples\RT\RTComm.llb` directory, on the Host PC.
7. Close all VIs and exit.

End of Activity 4-5.

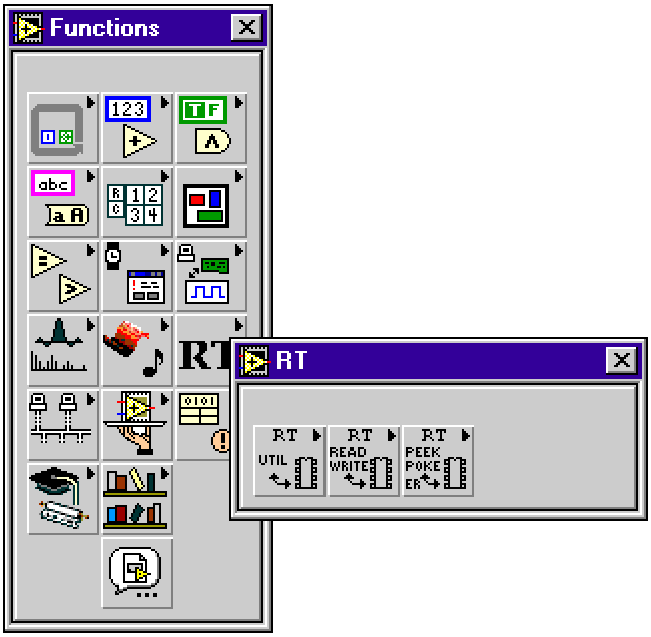
A LabVIEW RT Function Reference

This appendix contains function reference information for the LabVIEW RT VIs. In this appendix, VIs are arranged first by VI Library name, then by VI name.

Note In addition to the LabVIEW RT VIs, LabVIEW RT includes the PID Control for G Toolset. Because file I/O is not supported in LabVIEW RT, the Fuzzy Controller VI requires information for a specific designed fuzzy controller loaded from its corresponding data file (`*.fc`). You must save this information as the default values of the front panel controls of the Fuzzy Contoller VI. Complete documentation for the PID Control and Fuzzy Logic VIs is found in the PID Control for G Toolset documentation.

LabVIEW RT VI Library

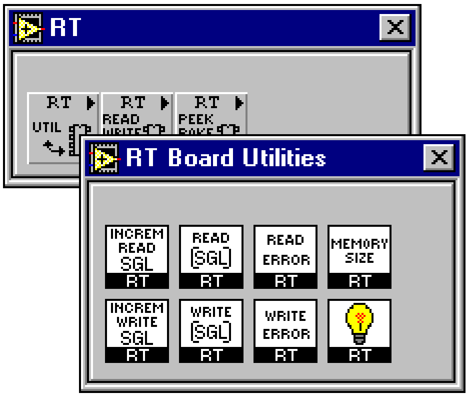
The LabVIEW RT VIs are available from the **Functions»RT** palette, shown below.



RT Board Utilities VIs

Use the RT Board Utilities VIs to check the shared memory size, toggle the board LEDs, and facilitate high level communication between the RT Series board and the host PC.

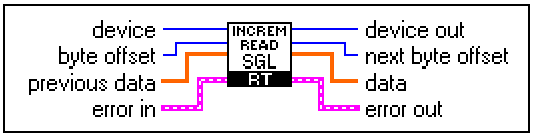
The RT Board Utilities VIs palette is shown below.



RT Incremental Single Read

Use the RT Incremental Single Read VI to share several data values running in continuous loops between the RT Engine and the RT Development System. This VI reads an array of singles data from shared memory, but only reads 16 bits of data each time it is called. The trade off is that it takes several loop iterations to read each single data value from memory.

Note This VI reads data written to shared memory by the RT Incremental Single Write VI, which skips writing new data if previous values have not been read, resulting in lost data points. See *RT Incremental Single Write* for more information.



I16 **device** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

U32 **byte offset** is the location in memory to start reading data.

[SGL] **previous data** is the previous data from the array of data read from shared memory.

error in describes error conditions that exist before this VI executes. If an error already occurred, the VI does not execute.

I16 **device out** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

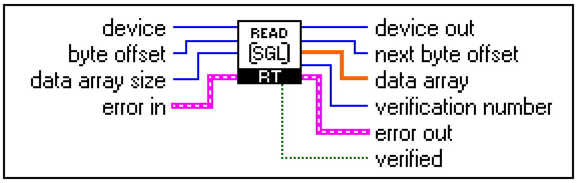
U32 **next byte offset** is the next available location in shared memory after this VI runs.

[SGL] **data** is an array of data read from shared memory.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

RT Read Single Array

Use the RT Read Single Array VI to read an array of data from shared memory.



I16 **device** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

U32 **byte offset** is the location in memory to start reading data.

I32 **data array size** is the number of values to read.

error in describes error conditions that exist before this VI executes. If an error already occurred, the VI does not execute.

I16 **device out** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

U32 **next byte offset** is the next available location in shared memory after this VI runs.

[SGL] **data array** is the array of data read from shared memory.

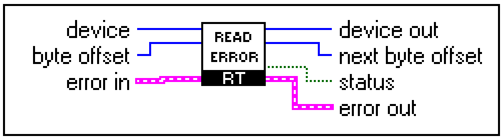
I16 **verification number** is written to memory by the RT Write Single Array VI and can be used to verify which array was read.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

TF **verified** indicates if an array was read.

RT Read Error From Memory

Use the RT Read Error From Memory VI to read the status and error code previously written to shared memory by the RT Write Error to Memory VI.



I16 **device** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

U32 **byte offset** is the location in memory to start reading data.

error in describes error conditions that exist before this VI executes. If an error already occurred, the VI does not execute.

I16 **device out** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

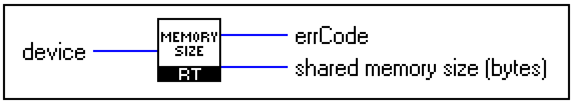
U32 **next byte offset** is the next available location in shared memory after this VI runs.

TF **status** is TRUE if an error occurred. If the status is TRUE, the VI does not perform any operations.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

RT Shared Memory Size

Use the RT Shared Memory Size VI to get the shared memory size of a PXI/PCI-7030 plug-in device. When running on the RT Engine, use device = 0 to get the size of the shared memory.



I16 **device** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

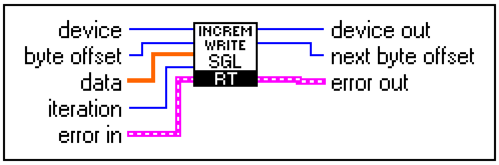
I32 **errCode** 0 = no error
errCode 37 = device not found

U32 **shared memory size (bytes)** is the size of shared memory, in bytes.

RT Incremental Single Write

Use the RT Incremental Single Write VI to share several data values between VIs running in the RT Engine and VIs running on the Host PC when your VIs are running in continuous loops and your loop speed needs to be as fast as possible.The RT Incremental Single Write VI writes an array of singles data to shared memory, but only writes 16 bits of data each time it is called. Because it only writes 16 bits of data each time it is called, this VI takes little time to execute, but may require several loop iterations to write each single data value to memory.

Note The RT Incremental Single Write VI only writes new data to shared memory after the previously written value has been read by RT incremental Single Read VI. The design of this subVI is not to write every value to memory, but rather to write the latest values to shared memory, which can result in lost data points.

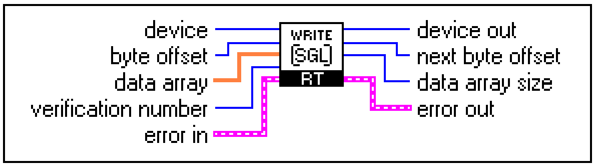


I16	**device** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.
U32	**byte offset** is the location in memory to start reading data.
[SGL]	**data** is the array of data read from shared memory.
I32	**iteration** is the current iteration of the loop. This parameter should be wired to the iteration terminal of the while loop in which this VI is used.
	error in describes error conditions that exist before this VI executes. If an error already occurred, the VI does not execute.
I16	**device out** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.
U32	**next byte offset** is the next available location in shared memory after this VI runs.
	error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

For examples using this VI, refer to the `One Channel PID - RT Engine.vi` and `One Channel PID - Host PC.vi`, located in the `LabVIEW\Examples\RT` directory.

RT Write Single Array

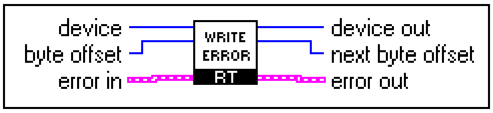
Use the RT Write Single Array VI to write a single array to shared memory.



I16	**device** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.
U32	**byte offset** is the location in memory to start reading data.
[SGL]	**data array** is the array of data read from shared memory.
I16	**verification number** is written before and after the array in shared memory and is used by the RT Read Single Array VI to verify the integrity of the read data.
	error in describes error conditions that exist before this VI executes. If an error already occurred, the VI does not execute.
I16	**device out** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.
U32	**next byte offset** is the next available location in shared memory after this VI runs.
I32	**data array size** is the number of values to read.
	error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

RT Write Error to Memory

Use the RT Write Error to Memory VI when you have a VI running on the RT Engine that needs to notify a VI running on the host PC that an error has occurred.



I16 **device** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

U32 **byte offset** is the location in memory to start reading data.

error in describes error conditions that exist before this VI executes. If an error already occurred, the VI does not execute.

I16 **device out** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

U32 **next byte** offset is the next available location in shared memory after this VI runs.

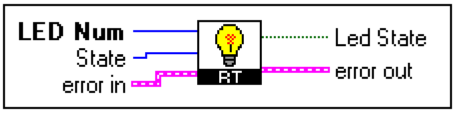
error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

This subVI should be placed outside a while loop, where it is called after the loop has finished. If the loop stops due to an error, this subVI is used to write that the error status and error code, but not the error source string, to shared memory.

RT Board LEDs

There are two LEDs, red and green, on the PCI and PXI boards. The red LED comes on any time the RT Engine attempts abnormal operations or encounters unhandled exception or the RT Engine terminates.

Use the RT Board LEDs VI to switch these LEDs on, off, or toggle their state.



LED Num
0 is the red LED
1 is the green LED

State
0 turns the LED off
1 turns the LED on
2 toggles the state of the LED

error in describes error conditions that exist before this VI executes. If an error already occurred, the VI does not execute.

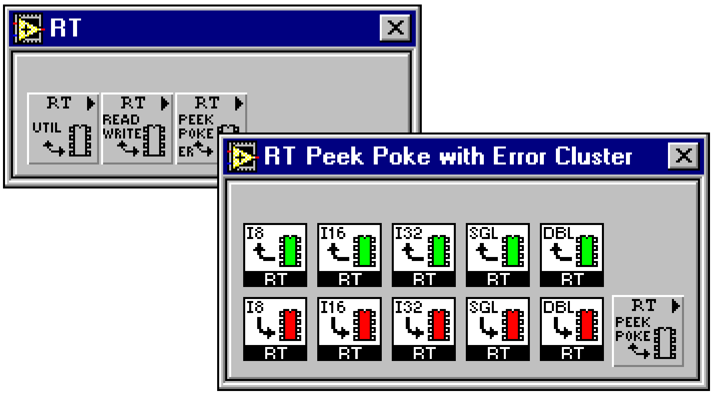
Led State indicates if the LED is on or off. The TRUE case indicates the LED is on.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

RT Peek Poke with Error Cluster VIs

Use the RT Peek Poke VIs to peek and poke data to and from the shared memory on the RT Series board. The RT Peek Poke VIs differ from the RT Low Level Peek Poke VIs in that they include the standard error cluster and provide the **next byte offset** indicator.

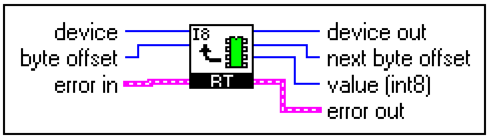
The RT Peek Poke with Error Cluster VIs palette is shown below.



RT Peek Byte

Use the RT Peek Byte VI to peek a byte (8 bit) number from the shared memory of your PXI/PCI-7030 RT Series board. To access shared memory, programs running on the RT Engine should use device = 0. Programs running on the host PC should use the device number assigned to the PCI/PXI-7030 processor board.

The RT Peek Byte VI differs from the RTLL Peek Byte VI in that RT Peek Byte uses the standard error cluster and provides the next byte offset indicator.



device is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

byte offset is the location in memory to start reading data.

error in describes error conditions that exist before this VI executes. If an error already occurred, the VI does not execute.

device out is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

next byte offset is the next available location in shared memory after this VI runs.

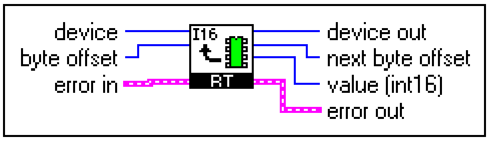
value (int8) is the number read from shared memory.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

RT Peek Word

Use the RT Peek Word VI to peek a word (16 bit) number from the shared memory of your PXI/PCI-7030 RT Series board. To access shared memory, programs running on the RT Engine should use device = 0. Programs running on the host PC should use the device number assigned to the PCI/PXI-7030 processor board.

The RT Peek Word VI differs from the RTLL Peek Word VI in that RT Peek Word uses the standard error cluster and provides the next byte offset indicator.



I16 **device** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

U32 **byte offset** is the location in memory to start reading data.

error in describes error conditions that exist before this VI executes. If an error already occurred, the VI does not execute.

I16 **device out** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

U32 **next byte offset** is the next available location in shared memory after this VI runs.

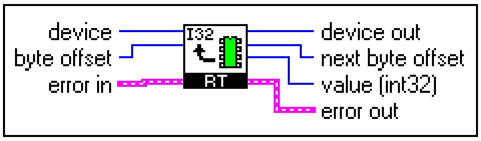
I16 **value (int16)** is the number read from shared memory.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

RT Peek Long

Use the RT Peek Long VI to peek a long (32 bit) number from the shared memory of your PXI/PCI-7030 RT Series board. To access shared memory, programs running on the RT Engine should use device = 0. Programs running on the host PC should use the device number assigned to the PCI/PXI-7030 processor board.

The RT Peek Long VI differs from the RTLL Peek Long VI in that RT Peek Long uses the standard error cluster and provides the next byte offset indicator.

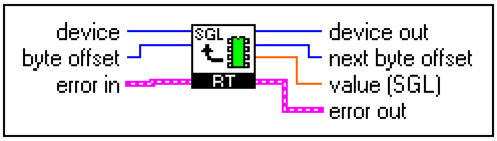


I16	**device** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.
U32	**byte offset** is the location in memory to start reading data.
	error in describes error conditions that exist before this VI executes. If an error already occurred, the VI does not execute.
I16	**device out** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.
U32	**next byte offset** is the next available location in shared memory after this VI runs.
I32	**value (int32)** is the number read from shared memory.
	error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

RT Peek Single

Use the RT Peek Single VI to peek a single precision (32 bit) floating point number from the shared memory of your PXI/PCI-7030 RT Series board. To access shared memory, programs running on the RT Engine should use device = 0. Programs running on the host PC should use the device number assigned to the PCI/PXI-7030 processor board.

The RT Peek Single VI differs from the RTLL Peek Single VI in that RT Peek Single uses the standard error cluster and provides the next byte offset indicator.



I16 **device** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

U32 **byte offset** is the location in memory to start reading data.

error in describes error conditions that exist before this VI executes. If an error already occurred, the VI does not execute.

I16 **device out** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

U32 **next byte offset** is the next available location in shared memory after this VI runs.

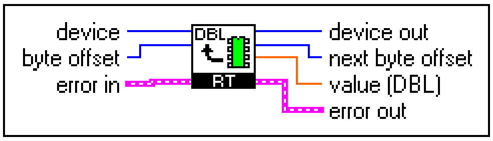
SGL **value (SGL)** is the number read from shared memory.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

RT Peek Double

Use the RT Peek Double VI to peek a double precision (64 bit) floating point number from the shared memory of your PXI/PCI-7030 RT Series board. To access shared memory, programs running on the RT Engine should use device = 0. Programs running on the host PC should use the device number assigned to the PCI/PXI-7030 processor board.

The RT Peek Double VI differs from the RTLL Peek Double VI in that RT Peek Double uses the standard error cluster and provides the next byte offset indicator.



device is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

byte offset is the location in memory to start reading data.

error in describes error conditions that exist before this VI executes. If an error already occurred, the VI does not execute.

device out is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

next byte offset is the next available location in shared memory after this VI runs.

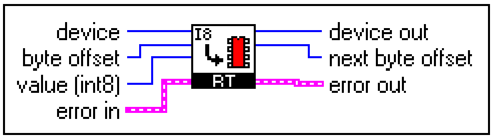
value (DBL) is the number read from shared memory.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

RT Poke Byte

Use the RT Poke Byte VI to poke a byte (8 bit) number to the shared memory of your PXI/PCI-7030 RT Series board. To access shared memory, programs running on the RT Engine should use device = 0. Programs running on the host PC should use the device number assigned to the PCI/PXI-7030 processor board.

The RT Poke Byte VI differs from the RTLL Poke Byte VI in that RT Peek Byte uses the standard error cluster and provides the next byte offset indicator.

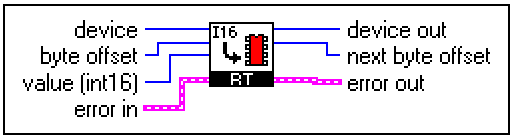


I16	**device** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.
U32	**byte offset** is the location in memory to start reading data.
I8	**value (int8)** is the number written to shared memory.
	error in describes error conditions that exist before this VI executes. If an error already occurred, the VI does not execute.
I16	**device out** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.
U32	**next byte offset** is the next available location in shared memory after this VI runs.
	error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

RT Poke Word

Use the RT Poke Word VI to poke a word (16 bit) number to the shared memory of your PXI/PCI-7030 RT Series board. To access shared memory, programs running on the RT Engine should use device = 0. Programs running on the host PC should use the device number assigned to the PCI/PXI-7030 processor board.

The RT Poke Word VI differs from the RTLL Poke Word VI in that RT Peek Word uses the standard error cluster and provides the next byte offset indicator.

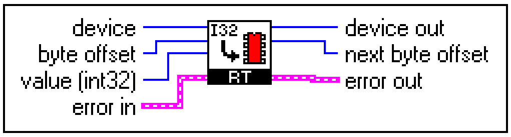


I16	**device** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.
U32	**byte offset** is the location in memory to start reading data.
I16	**value (int16)** is the number written to shared memory.
	error in describes error conditions that exist before this VI executes. If an error already occurred, the VI does not execute.
I16	**device out** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.
U32	**next byte offset** is the next available location in shared memory after this VI runs.
	error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

RT Poke Long

Use the RT Poke Long VI to poke a long (32 bit) number to the shared memory of your PXI/PCI-7030 RT Series board. To access shared memory, programs running on the RT Engine should use device = 0. Programs running on the host PC should use the device number assigned to the PCI/PXI-7030 processor board.

The RT Poke Long VI differs from the RTLL Poke Long VI in that RT Peek Long uses the standard error cluster and provides the next byte offset indicator.



I16 **device** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

U32 **byte offset** is the location in memory to start reading data.

I32 **value (int32)** is the number written to shared memory.

error in describes error conditions that exist before this VI executes. If an error already occurred, the VI does not execute.

I16 **device out** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

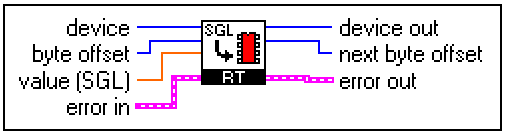
U32 **next byte offset** is the next available location in shared memory after this VI runs.

error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

RT Poke Single

Use the RT Poke Single VI to poke a single precision (32 bit) floating point number to the shared memory of your PXI/PCI-7030 RT Series board. To access shared memory, programs running on the RT Engine should use device = 0. Programs running on the host PC should use the device number assigned to the PCI/PXI-7030 processor board.

The RT Poke Single VI differs from the RTLL Poke Single VI in that RT Peek Single uses the standard error cluster and provides the next byte offset indicator.

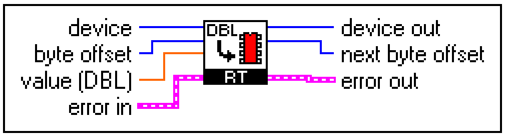


I16	**device** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.
U32	**byte offset** is the location in memory to start reading data.
SGL	**value (SGL)** is the number written to shared memory.
	error in describes error conditions that exist before this VI executes. If an error already occurred, the VI does not execute.
I16	**device out** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.
U32	**next byte offset** is the next available location in shared memory after this VI runs.
	error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

RT Poke Double

Use the RT Poke Double VI to poke a double precision (64 bit) floating point number to the shared memory of your PXI/PCI-7030 RT Series board. To access shared memory, programs running on the RT Engine should use device = 0. Programs running on the host PC should use the device number assigned to the PCI/PXI-7030 processor board.

The RT Poke Double VI differs from the RTLL Poke Double VI in that RT Peek Double uses the standard error cluster and provides the next byte offset indicator.

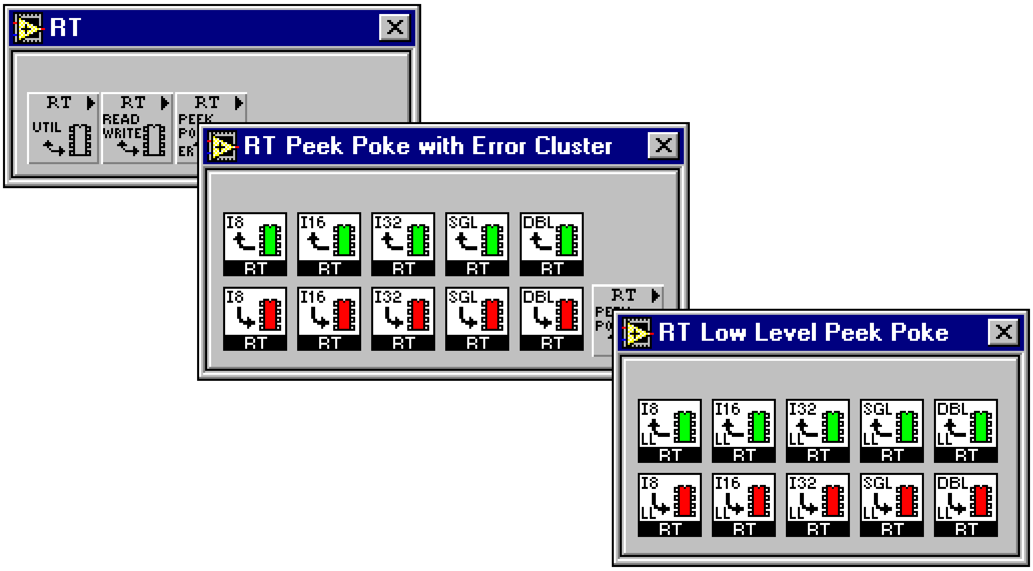


I16	**device** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.
U32	**byte offset** is the location in memory to start reading data.
DBL	**value (DBL)** is the number written to shared memory.
	error in describes error conditions that exist before this VI executes. If an error already occurred, the VI does not execute.
I16	**device out** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.
U32	**next byte offset** is the next available location in shared memory after this VI runs.
	error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

RT Low Level Peek Poke VIs

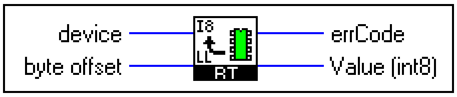
Use the RT Low Level Peek Poke VIs to peek and poke data to and from the shared memory on the RT Series board. The RT Low Level Peek Poke VIs differ from the RT Peek Poke with Error Cluster VIs in that they do not include the standard error cluster nor provide the **next byte offset** indicator.

The RT Low Level Peek Poke VIs palette is shown below.



RTLL Peek Byte

Use the RTLL Peek Byte VI to peek a byte (8 bit number) from the shared memory of your PXI/PCI-7030 RT Series board. To access shared memory, programs running on the RT Engine should use device = 0. Programs running on the host PC should use the device number assigned to the PCI/PXI-7030 processor board.



I16 **device** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

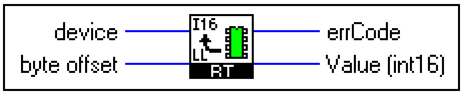
U32 **byte offset** is the location in memory to start reading data.

I32 **errCode** 0 = no error
errCode 37 = device not found
errCode 3 = offset is outside the memory window

I8 **Value (int8)** is the number read from shared memory.

RTLL Peek Word

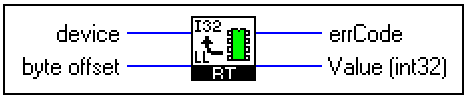
Use the RTLL Peek Word VI to peek a word (16 bit number) from the shared memory of your PXI/PCI-7030 RT Series board. To access shared memory, programs running on the RT Engine should use device = 0. Programs running on the host PC should use the device number assigned to the PCI/PXI-7030 processor board.



I16	**device** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.
U32	**byte offset** is the location in memory to start reading data.
I32	**errCode** 0 = no error **errCode** 37 = device not found **errCode** 3 = offset is outside the memory window
I16	**Value (int16)** is the number read from shared memory.

RTLL Peek Long

Use the RTLL Peek Long VI to peek a long (32 bit number) from the shared memory of your PXI/PCI-7030 RT Series board. To access shared memory, programs running on the RT Engine should use device = 0. Programs running on the host PC should use the device number assigned to the PCI/PXI-7030 processor board.



I16 **device** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

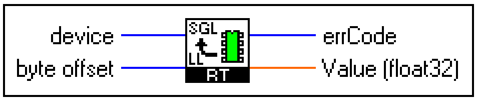
U32 **byte offset** is the location in memory to start reading data.

I32 **errCode** 0 = no error
errCode 37 = device not found
errCode 3 = offset is outside the memory window

I32 **Value (int32)** is the number read from shared memory.

RTLL Peek Single

Use the RTLL Peek Single VI to peek single precision floating point number (32-bit) from the shared memory of your PXI/PCI-7030 RT Series board. To access shared memory, programs running on the RT Engine should use device = 0. Programs running on the host PC should use the device number assigned to the PCI/PXI-7030 processor board.



I16 **device** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

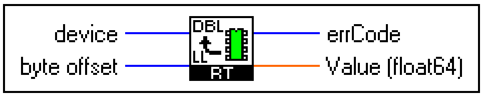
U32 **byte offset** is the location in memory to start reading data.

I32 **errCode** 0 = no error
errCode 37 = device not found
errCode 3 = offset is outside the memory window

SGL **Value (float32)** is the number read from shared memory.

RTLL Peek Double

Use the RTLL Peek Double VI to peek double precision floating point number (64 bit) from the shared access of your PXI/PCI-7030 RT Series board. To access shared memory, programs running on the RT Engine should use device = 0. Programs running on the host PC should use the device number assigned to the PCI/PXI-7030 processor board.



I16 **device** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

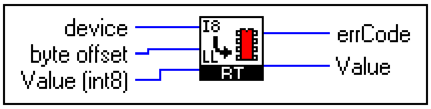
U32 **byte offset** is the location in memory to start reading data.

I32 **errCode** 0 = no error
errCode 37 = device not found
errCode 3 = offset is outside the memory window

DBL **Value (float64)** is the number read from shared memory.

RTLL Poke Byte

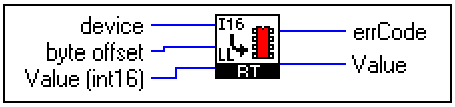
Use the RTLL Poke Byte VI to poke a byte (8 bit) number to the shared memory of your PXI/PCI-7030 RT Series board. To access shared memory, programs running on the RT Engine should use device = 0. Programs running on the host PC should use the device number assigned to the PCI/PXI-7030 processor board.



I16	**device** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.
U32	**byte offset** is the location in memory to start reading data.
I8	**Value (int8)** is the number written to shared memory.
I32	**errCode** 0 = no error **errCode** 37 = device not found **errCode** 3 = offset is outside the memory window
U8	**Value** is the number written to shared memory.

RTLL Poke Word

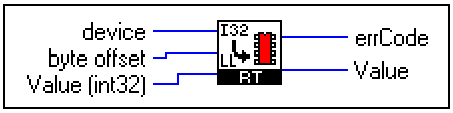
Use the RTLL Poke Word VI to poke a word (16 bit) number to the shared memory of your PXI/PCI-7030 RT Series board. To access shared memory, programs running on the RT Engine should use device = 0. Programs running on the host PC should use the device number assigned to the PCI/PXI-7030 processor board7030 processor board.



I16	**device** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.
U32	**byte offset** is the location in memory to start reading data.
I16	**Value (int16)** is the number written to shared memory.
I32	**errCode** 0 = no error **errCode** 37 = device not found **errCode** 3 = offset is outside the memory window
I16	**Value** is the number written to shared memory.

RTLL Poke Long

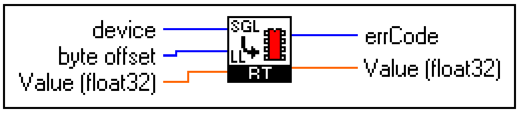
Use the RTLL Poke Long VI to poke a long (32 bit) number to the shared memory of your PXI/PCI-7030 RT Series board. To access shared memory, programs running on the RT Engine should use device = 0. Programs running on the host PC should use the device number assigned to the PCI/PXI-7030 processor board.



I16	**device** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.
U32	**byte offset** is the location in memory to start reading data.
I32	**Value (int32)** is the number written to shared memory.
I32	**errCode** 0 = no error **errCode** 37 = device not found **errCode** 3 = offset is outside the memory window
I32	**Value** is the number written to shared memory.

RTLL Poke Single

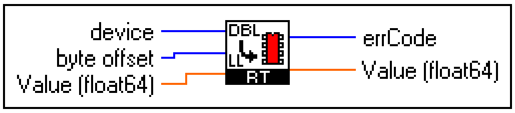
Use the RTLL Poke Single VI to poke a single precision (32 bit) floating point number to the shared memory of your PXI/PCI-7030 RT Series board. To access shared memory, programs running on the RT Engine should use device = 0. Programs running on the host PC should use the device number assigned to the PCI/PXI-7030 processor board.



I16	**device** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.
U32	**byte offset** is the location in memory to start reading data.
SGL	**Value (float32)** is the number written to shared memory.
I32	**errCode** 0 = no error **errCode** 37 = device not found **errCode** 3 = offset is outside the memory window
SGL	**Value (float32)** is the number written to shared memory.

RTLL Poke Double

Use the RTLL Poke Double VI to poke a double precision (64 bit) floating point number to the shared memory of your PXI/PCI-7030 RT Series board. To access shared memory, programs running on the RT Engine should use device = 0. Programs running on the host PC should use the device number assigned to the PCI/PXI-7030 processor board.



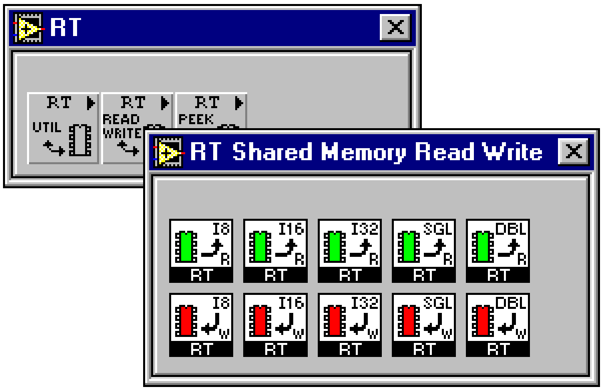
I16	**device** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.
U32	**byte offset** is the location in memory to start reading data.
DBL	**Value (float64)** is the number written to shared memory.
I32	**errCode** 0 = no error **errCode** 37 = device not found **errCode** 3 = offset is outside the memory window
DBL	**Value (float64)** is the number written to shared memory.

RT Shared Memory Read Write VIs

Use the RT Read Write VIs read and write data from the shared memory of your PXI/PCI-7030 RT Series board. The read and write shared memory VIs differ from the peek and poke shared memory VIs in that the read and write shared memory VIs use a read/write flag to determine if data has changed.

When used in a loop, a Write Shared Memory VI first checks to see if its new value is different from the previous value it wrote to memory. If the new value equals the previous value, the data is not written to shared memory. If the new value is different, the read/write flag is checked to see if the last value written to shared memory has been read. If it has (read/write flag = 0), the new value is written to shared memory and the read/write flag is set to 1. If it has not been read, the new value is not written to shared memory.

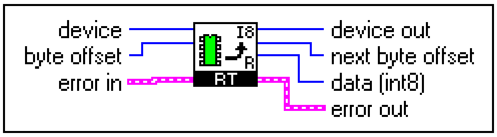
The RT Shared Memory Read Write VIs palette is shown below.



RT Read Byte

Use the RT Read Byte VI to read a byte from the shared memory of your PXI/PCI-7030 RT Series board. To access shared memory, programs running on the RT Engine should use device = 0. Programs running on the host PC should use the device number assigned to the PCI/PXI-7030 processor board.

The read and write shared memory VIs differ from the peek and poke shared memory VIs in that the read and write VIs use a read/write flag to determine if data has changed.



I16 **device** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

U32 **byte offset** is the location in memory to start reading data.

error in describes error conditions that exist before this VI executes. If an error already occurred, the VI does not execute.

I16 **device out** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

U32 **next byte offset** is the next available location in shared memory after this VI runs.

I8 **data (int8)** is the data read from shared memory.

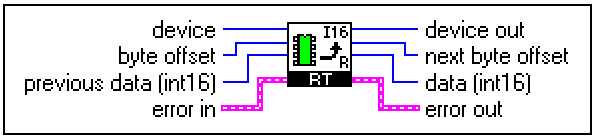
error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

When used in a loop, a Write Shared Memory VI first checks to see if its new value is different from the previous value it wrote to memory. If the new value equals the previous value, the data is not written to shared memory. If the new value is different, the read/write flag is checked to see if the last value written to shared memory has been read. If it has (read/write flag = 0), the new value is written to shared memory and the read/write flag is set to 1. If it has not been read, the new value is not written to shared memory.

RT Read Word

Use the RT Read Word VI to read a word (16 bit) number from the shared memory of your PXI/PCI-7030 RT Series board. To access shared memory, programs running on the RT Engine should use device = 0. Programs running on the host PC should use the device number assigned to the PCI/PXI-7030 processor board.

The read and write shared memory VIs differ from the peek and poke shared memory VIs in that the read and write VIs use a read/write flag to determine if data has changed.



device is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

byte offset is the location in memory to start reading data.

previous data (int16) is the previous data from the array of data read from shared memory. This parameter should be wired from a shift register on the left side of the while loop in which this VI is used. Doing this assures that if a data value to be written has not changed, no write occurs.

error in describes error conditions that exist before this VI executes. If an error already occurred, the VI does not execute.

device out is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

next byte offset is the next available location in shared memory after this VI runs.

data (int16) is the data read from shared memory.

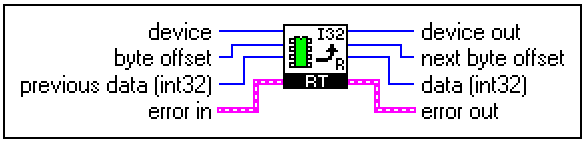
error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

When used in a loop, a Write Shared Memory VI first checks to see if its new value is different from the previous value it wrote to memory. If the new value equals the previous value, the data is not written to shared memory. If the new value is different, the read/write flag is checked to see if the last value written to shared memory has been read. If it has (read/write flag = 0), the new value is written to shared memory and the read/write flag is set to 1. If it has not been read, the new value is not written to shared memory.

RT Read Long

Use the RT Read Long VI to read a long (32 bit) number from the shared memory of your PXI/PCI-7030 RT Series board. To access shared memory, programs running on the RT Engine should use device = 0. Programs running on the host PC should use the device number assigned to the PCI/PXI-7030 processor board.

The read and write shared memory VIs differ from the peek and poke shared memory VIs in that the read and write VIs use a read/write flag to determine if data has changed.



I16 **device** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

U32 **byte offset** is the location in memory to start reading data.

I32 **previous data (int32)** is the previous data from the array of data read from shared memory. This parameter should be wired from a shift register on the left side of the while loop in which this VI is used. Doing this assures that if a data value to be written has not changed, no write occurs.

error in describes error conditions that exist before this VI executes. If an error already occurred, the VI does not execute.

I16 **device out** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

U32 **next byte offset** is the next available location in shared memory after this VI runs.

I32 **data (int32)** is the data read from shared memory.

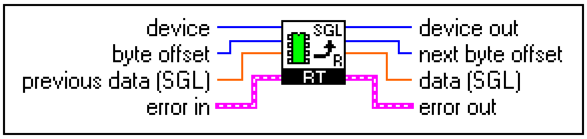
error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

When used in a loop, a Write Shared Memory VI first checks to see if its new value is different from the previous value it wrote to memory. If the new value equals the previous value, the data is not written to shared memory. If the new value is different, the read/write flag is checked to see if the last value written to shared memory has been read. If it has (read/write flag = 0), the new value is written to shared memory and the read/write flag is set to 1. If it has not been read, the new value is not written to shared memory.

RT Read Single

Use the RT Read Single VI to read a single precision (32 bit) floating point number from the shared memory of your PXI/PCI-7030 RT Series board. To access shared memory, programs running on the RT Engine should use device = 0. Programs running on the host PC should use the device number assigned to the PCI/PXI-7030 processor board.

The read and write shared memory VIs differ from the peek and poke shared memory VIs in that the read and write VIs use a read/write flag to determine if data has changed.



I16 **device** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

U32 **byte offset** is the location in memory to start reading data.

SGL **previous data (SGL)** is the previous data from the array of data read from shared memory. This parameter should be wired from a shift register on the left side of the while loop in which this VI is used. Doing this assures that if a data value to be written has not changed, no write occurs.

error in describes error conditions that exist before this VI executes. If an error already occurred, the VI does not execute.

I16 **device out** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

U32 **next byte offset** is the next available location in shared memory after this VI runs.

SGL **data (SGL)** is the data read from shared memory.

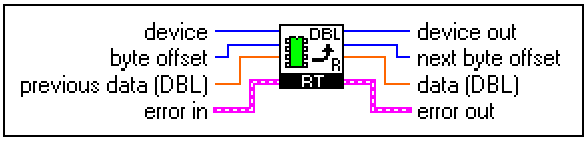
error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

When used in a loop, a Write Shared Memory VI first checks to see if its new value is different from the previous value it wrote to memory. If the new value equals the previous value, the data is not written to shared memory. If the new value is different, the read/write flag is checked to see if the last value written to shared memory has been read. If it has (read/write flag = 0), the new value is written to shared memory and the read/write flag is set to 1. If it has not been read, the new value is not written to shared memory.

RT Read Double

Use the RT Read Double VI to read a double precision (64 bit) floating point number from the shared memory of your PXI/PCI-7030 RT Series board. To access shared memory, programs running on the RT Engine should use device = 0. Programs running on the host PC should use the device number assigned to the PCI/PXI-7030 processor board.

The read and write shared memory VIs differ from the peek and poke shared memory VIs in that the read and write VIs use a read/write flag to determine if data has changed.



I16 **device** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

U32 **byte offset** is the location in memory to start reading data.

DBL **previous data (DBL)** is the previous data from the array of data read from shared memory. This parameter should be wired from a shift register on the left side of the while loop in which this VI is used. Doing this assures that if a data value to be written has not changed, no write occurs.

error in describes error conditions that exist before this VI executes. If an error already occurred, the VI does not execute.

I16 **device out** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

U32 **next byte offset** is the next available location in shared memory after this VI runs.

DBL **data (DBL)** is the data read from shared memory.

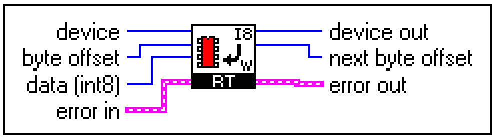
error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

When used in a loop, a Write Shared Memory VI first checks to see if its new value is different from the previous value it wrote to memory. If the new value equals the previous value, the data is not written to shared memory. If the new value is different, the read/write flag is checked to see if the last value written to shared memory has been read. If it has (read/write flag = 0), the new value is written to shared memory and the read/write flag is set to 1. If it has not been read, the new value is not written to shared memory.

RT Write Byte

Use the RT Write Byte VI to write a byte (8 bit) number to the shared memory of your PXI/PCI-7030 RT Series board. To access shared memory, programs running on the RT Engine should use device = 0. Programs running on the host PC should use the device number assigned to the PCI/PXI-7030 processor board.

The read and write shared memory VIs differ from the peek and poke shared memory VIs in that the read and write VIs use a read/write flag to determine if data has changed.



I16 **device** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

U32 **byte offset** is the location in memory to start reading data.

I8 **data (int8)** is the data written to shared memory.

error in describes error conditions that exist before this VI executes. If an error already occurred, the VI does not execute.

I16 **device out** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

U32 **next byte offset** is the next available location in shared memory after this VI runs.

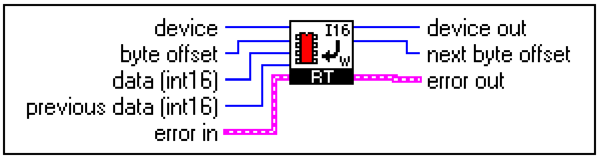
error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

When used in a loop, a Write Shared Memory VI first checks to see if its new value is different from the previous value it wrote to memory. If the new value equals the previous value, the data is not written to shared memory. If the new value is different, the read/write flag is checked to see if the last value written to shared memory has been read. If it has (read/write flag = 0), the new value is written to shared memory and the read/write flag is set to 1. If it has not been read, the new value is not written to shared memory.

RT Write Word

Use the RT Write Word VI to write a word (16 bit) number to the shared memory of your PXI/PCI-7030 RT Series board. To access shared memory, programs running on the RT Engine should use device = 0. Programs running on the host PC should use the device number assigned to the PCI/PXI-7030 processor board.

The read and write shared memory VIs differ from the peek and poke shared memory VIs in that the read and write VIs use a read/write flag to determine if data has changed.



I16 **device** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

U32 **byte offset** is the location in memory to start reading data.

I16 **data (int16)** is the data written to shared memory.

I16 **previous data (int16)** is the previous data from the array of data read from shared memory. This parameter should be wired from a shift register on the left side of the while loop in which this VI is used. Doing this assures that if a data value to be written has not changed, no write occurs.

error in describes error conditions that exist before this VI executes. If an error already occurred, the VI does not execute.

I16 **device out** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

U32 **next byte offset** is the next available location in shared memory after this VI runs.

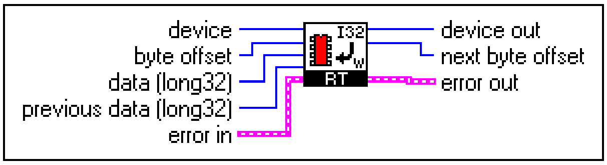
error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

When used in a loop, a Write Shared Memory VI first checks to see if its new value is different from the previous value it wrote to memory. If the new value equals the previous value, the data is not written to shared memory. If the new value is different, the read/write flag is checked to see if the last value written to shared memory has been read. If it has (read/write flag = 0), the new value is written to shared memory and the read/write flag is set to 1. If it has not been read, the new value is not written to shared memory.

RT Write Long

Use the RT Write Long VI to write a long (32 bit) number to the shared memory of your PXI/PCI-7030 RT Series board. To access shared memory, programs running on the RT Engine should use device = 0. Programs running on the host PC should use the device number assigned to the PCI/PXI-7030 processor board.

The read and write shared memory VIs differ from the peek and poke shared memory VIs in that the read and write VIs use a read/write flag to determine if data has changed.



I16 **device** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

U32 **byte offset** is the location in memory to start reading data.

I32 **data (long32)** is the data written to shared memory.

I32 **previous data (long32)** is the previous data from the array of data read from shared memory. This parameter should be wired from a shift register on the left side of the while loop in which this VI is used. Doing this assures that if a data value to be written has not changed, no write occurs.

error in describes error conditions that exist before this VI executes. If an error already occurred, the VI does not execute.

I16 **device out** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.

U32 **next byte offset** is the next available location in shared memory after this VI runs.

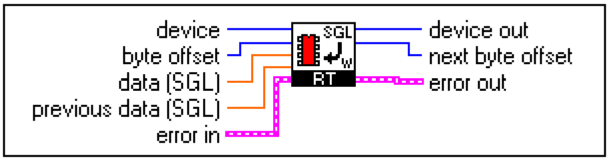
error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

When used in a loop, a Write Shared Memory VI first checks to see if its new value is different from the previous value it wrote to memory. If the new value equals the previous value, the data is not written to shared memory. If the new value is different, the read/write flag is checked to see if the last value written to shared memory has been read. If it has (read/write flag = 0), the new value is written to shared memory and the read/write flag is set to 1. If it has not been read, the new value is not written to shared memory.

RT Write Single

Use the RT Write Single VI to write a single precision (32 bit) floating point number to the shared memory of your PXI/PCI-7030 RT Series board. To access shared memory, programs running on the RT Engine should use device = 0. Programs running on the host PC should use the device number assigned to the PCI/PXI-7030 processor board.

The read and write shared memory VIs differ from the peek and poke shared memory VIs in that the read and write VIs use a read/write flag to determine if data has changed.



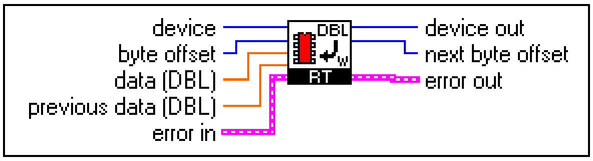
I16	**device** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.
U32	**byte offset** is the location in memory to start reading data.
SGL	**data (SGL)** is the data written to shared memory.
SGL	**previous data (SGL)** is the previous data from the array of data read from shared memory. This parameter should be wired from a shift register on the left side of the while loop in which this VI is used. Doing this assures that if a data value to be written has not changed, no write occurs.
	error in describes error conditions that exist before this VI executes. If an error already occurred, the VI does not execute.
I16	**device out** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.
U32	**next byte offset** is the next available location in shared memory after this VI runs.
	error out contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

When used in a loop, a Write Shared Memory VI first checks to see if its new value is different from the previous value it wrote to memory. If the new value equals the previous value, the data is not written to shared memory. If the new value is different, the read/write flag is checked to see if the last value written to shared memory has been read. If it has (read/write flag = 0), the new value is written to shared memory and the read/write flag is set to 1. If it has not been read, the new value is not written to shared memory.

RT Write Double

Use the RT Write Double VI to write a double precision (32 bit) floating point number to the shared memory of your PXI/PCI-7030 RT Series board. To access shared memory, programs running on the RT Engine should use device = 0. Programs running on the host PC should use the device number assigned to the PCI/PXI-7030 processor board.

The read and write shared memory VIs differ from the peek and poke shared memory VIs in that the read and write VIs use a read/write flag to determine if data has changed.



I16	**device** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.
U32	**byte offset** is the location in memory to start reading data.
DBL	**data (DBL)** is the data written to shared memory.
DBL	**previous data (DBL)** is the previous data from the array of data read from shared memory. This parameter should be wired from a shift register on the left side of the while loop in which this VI is used. Doing this assures that if a data value to be written has not changed, no write occurs.
▨	**error in** describes error conditions that exist before this VI executes. If an error already occurred, the VI does not execute.
I16	**device out** is the device number you assigned to the RT Series processor board during configuration. This parameter defaults to 1.
U32	**next byte offset** is the next available location in shared memory after this VI runs.
▨	**error out** contains error information. If the **error in** cluster indicated an error, the **error out** cluster contains the same information. Otherwise, **error out** describes the error status of this VI.

When used in a loop, a Write Shared Memory VI first checks to see if its new value is different from the previous value it wrote to memory. If the new value equals the previous value, the data is not written to shared memory. If the new value is different, the read/write flag is checked to see if the last value written to shared memory has been read. If it has (read/write flag = 0), the new value is written to shared memory and the read/write flag is set to 1. If it has not been read, the new value is not written to shared memory.

B Specifications

This appendix lists the specifications of the embedded processor system.

Processor

Processor	AMD 486 DX5 32-Bit Architecture
Processor Clock Speed	133 MHz
CPU Bus Speed	33 MHz
Memory	8 MB DRAM user programmable 60 ns, EDO, 5V, 72-PinSODIMM
On-Chip Cache	16 KB Write-Back
L2 Cache	256 KB Write-Back
Floating Point Unit	Yes

Host Embedded Communication

Shared Memory

Type	SRAM
Size	1 KB user available

Bus Interface

Type	PCI Slave

Power Requirement

PCI-7030 only (without daughter board)	1.9 A at +5 VDC (±5%)
PXI-7030 only (without daughter board)	2.0 A at +5 VDC (±5%)

Note To calculate the total power requirement, add the processor board requirement (shown above) to the I/O board requirement given in the I/O board manual.

Physical

Dimensions (not including connectors)

PCI-7030	312.9 by 160.5 cm (12.3 by 4.2 in) One PCI slot
PXI-7030	16 by 10 cm (6.3 by 3.9 in) Two PXI slots

Environment

Operating Temperature	0 ° to 55 °C
Storage Temperature	–20 ° to 70 °C
Relative Humidity	10% to 90% noncondensing

RTSI

Refer to the *RTSI* section of Chapter 3, *Hardware Overview*.

I/O Daughter Board Specifications

See appropriate I/O board manual, listed in the following table.

RT Series Board	DAQ Manual
PXI-7030/6040E	*PXI E Series User Manual*
PXI-7030/6030E	*PXI E Series User Manual*
PXI-7030/6533	*DIO 6533 User Manual*
PCI-7030/6040E	*PCI E Series User Manual*
PCI-7030/6030E	*PCI E Series User Manual*
PCI-7030/6533	*DIO 6533 User Manual*

C

Technical Support Resources

This appendix describes the comprehensive resources available to you in the Technical Support section of the National Instruments Web site and provides technical support telephone numbers for you to use if you have trouble connecting to our Web site or if you do not have internet access.

NI Web Support

To provide you with immediate answers and solutions 24 hours a day, 365 days a year, National Instruments maintains extensive online technical support resources. They are available to you at no cost, are updated daily, and can be found in the Technical Support section of our Web site at `www.natinst.com/support`.

Online Problem-Solving and Diagnostic Resources

- **KnowledgeBase**—A searchable database containing thousands of frequently asked questions (FAQs) and their corresponding answers or solutions, including special sections devoted to our newest products. The database is updated daily in response to new customer experiences and feedback.
- **Troubleshooting Wizards**—Step-by-step guides lead you through common problems and answer questions about our entire product line. Wizards include screen shots that illustrate the steps being described and provide detailed information ranging from simple getting started instructions to advanced topics.
- **Product Manuals**—A comprehensive, searchable library of the latest editions of National Instruments hardware and software product manuals.
- **Hardware Reference Database**—A searchable database containing brief hardware descriptions, mechanical drawings, and helpful images of jumper settings and connector pinouts.
- **Application Notes**—A library with more than 100 short papers addressing specific topics such as creating and calling DLLs, developing your own instrument driver software, and porting applications between platforms and operating systems.

Software-Related Resources

- **Instrument Driver Network**—A library with hundreds of instrument drivers for control of standalone instruments via GPIB, VXI, or serial interfaces. You also can submit a request for a particular instrument driver if it does not already appear in the library.
- **Example Programs Database**—A database with numerous, non-shipping example programs for National Instruments programming environments. You can use them to complement the example programs that are already included with National Instruments products.
- **Software Library**—A library with updates and patches to application software, links to the latest versions of driver software for National Instruments hardware products, and utility routines.

Worldwide Support

National Instruments has offices located around the globe. Many branch offices maintain a Web site to provide information on local services. You can access these Web sites from `www.natinst.com/worldwide`.

If you have trouble connecting to our Web site, please contact your local National Instruments office or the source from which you purchased your National Instruments product(s) to obtain support.

For telephone support in the United States, dial 512 795 8248. For telephone support outside the United States, contact your local branch office:

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20,
Brazil 011 284 5011, Canada (Ontario) 905 785 0085,
Canada (Québec) 514 694 8521, China 0755 3904939,
Denmark 45 76 26 00, Finland 09 725 725 11, France 01 48 14 24 24,
Germany 089 741 31 30, Hong Kong 2645 3186, India 91805275406,
Israel 03 6120092, Italy 02 413091, Japan 03 5472 2970,
Korea 02 596 7456, Mexico (D.F.) 5 280 7625,
Mexico (Monterrey) 8 357 7695, Netherlands 0348 433466,
Norway 32 27 73 00, Singapore 2265886, Spain (Madrid) 91 640 0085,
Spain (Barcelona) 93 582 0251, Sweden 08 587 895 00,
Switzerland 056 200 51 51, Taiwan 02 2377 1200,
United Kingdom 01635 523545

Glossary

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

Numbers/Symbols

%	percent
+	positive of, or plus
–	negative of, or minus
°	degree
W	ohm

A

A	amperes
AC	alternating current
AC coupled	allowing the transmission of AC signals while blocking DC signals
A/D	analog-to-digital
address	character code that identifies a specific location (or series of locations) in memory
alias	a false lower frequency component that appears in sampled data acquired at too low a sampling rate

ALU	arithmetic logic unit—the element(s) in a processing system that perform(s) the mathematical functions such as addition, subtraction, multiplication, division, inversion, AND, OR, NAND, and NOR
amplification	a type of signal conditioning that improves accuracy in the resulting digitized signal and reduces noise
ASIC	Application-Specific Integrated Circuit—a proprietary semiconductor component designed and manufactured to perform a set of specific functions for a specific customer
asynchronous	(1) hardware—a property of an event that occurs at an arbitrary time, without synchronization to a reference clock (2) software—a property of a function that begins an operation and returns prior to the completion or termination of the operation

B

b	bit—one binary digit, either 0 or 1
B	byte—eight related bits of data, an eight-bit binary number. Also used to denote the amount of memory required to store one byte of data.
bandwidth	the range of frequencies present in a signal, or the range of frequencies to which a measuring device can respond
base address	a memory address that serves as the starting address for programmable registers. All other addresses are located by adding to the base address.
binary	a number system with a base of 2
BIOS	basic input/output system—BIOS functions are the fundamental level of any PC or compatible computer. BIOS functions embody the basic operations needed for successful use of the computer’s hardware resources.
buffer	temporary storage for acquired or generated data (software)
burst-mode	a high-speed data transfer in which the address of the data is sent followed by back-to-back data words while a physical signal is asserted
bus	the group of conductors that interconnect individual circuitry in a computer. Typically, a bus is the expansion vehicle to which I/O or other devices are connected. Examples of PC buses are the ISA and PCI bus.

C

C	Celsius
cache	high-speed processor memory that buffers commonly used instructions or data to increase processing throughput
channel	pin or wire lead to which you apply or from which you read the analog or digital signal. Analog signals can be single-ended or differential. For digital signals, you group channels to form ports. Ports usually consist of either four or eight digital channels.
channel clock	the clock controlling the time interval between individual channel sampling within a scan. Boards with simultaneous sampling do not have this clock.
clock	hardware component that controls timing for reading from or writing to groups
code width	the smallest detectable change in an input voltage of a DAQ device
counter/timer	a circuit that counts external pulses or clock pulses (timing)
coupling	the manner in which a signal is connected from one location to another
CPU	central processing unit
current drive capability	the amount of current a digital or analog output channel is capable of sourcing or sinking while still operating within voltage range specifications

D

D/A	digital-to-analog
DAC	digital-to-analog converter—an electronic device, often an integrated circuit, that converts a digital number into a corresponding analog voltage or current
daisy-chain	a method of propagating signals along a bus, in which the devices are prioritized on the basis of their position on the bus

DAQ	data acquisition—(1) collecting and measuring electrical signals from sensors, transducers, and test probes or fixtures and inputting them to a computer for processing; (2) collecting and measuring the same kinds of electrical signals with A/D and/or DIO boards plugged into a computer, and possibly generating control signals with D/A and/or DIO boards in the same computer
dB	decibel—the unit for expressing a logarithmic measure of the ratio of two signal levels: dB=20log10 V1/V2, for signals in volts
DC	direct current
DC coupled	allowing the transmission of both AC and DC signals
default setting	a default parameter value recorded in the driver. In many cases, the default input of a control is a certain value (often 0) that means *use the current default setting*. For example, the default input for a parameter may be *do not change current setting*, and the default setting may be *no AMUX-64T boards*. If you do change the value of such a parameter, the new value becomes the new setting. You can set default settings for some parameters in the configuration utility or manually using switches located on the device.
device	a plug-in data acquisition board, card, or pad that can contain multiple channels and conversion devices. Plug-in boards, PCMCIA cards, and devices such as the DAQPad-1200, which connects to your computer parallel port, are all examples of DAQ devices. SCXI modules are distinct from devices, with the exception of the SCXI-1200, which is a hybrid.
DIFF	differential mode
digital port	*See* port.
digital trigger	a TTL level signal having two discrete levels—a high and a low level
DIO	digital input/output
DIP	dual inline package
dithering	the addition of Gaussian noise to an analog input signal
DMA	direct memory access—a method by which data can be transferred to/from computer memory from/to a device or memory on the bus while the processor does something else. DMA is the fastest method of transferring data to/from computer memory.

DOS	disk operating system
DRAM	dynamic RAM
drivers	software that controls a specific hardware device such as a DAQ board or a GPIB interface board
dual-access memory	*See* shared memory.

E

ECL	emitter-coupled logic
EEPROM	electrically erasable programmable read-only memory—ROM that can be erased with an electrical signal and reprogrammed
EMC	electromechanical compliance
encoder	a device that converts linear or rotary displacement into digital or pulse signals. The most popular type of encoder is the optical encoder, which uses a rotating disk with alternating opaque areas, a light source, and a photodetector.
ETS	equivalent-time sampling
event	the condition or state of an analog or digital signal
expansion ROM	an onboard EEPROM that may contain device-specific initialization and system boot functionality
external trigger	a voltage pulse from an external source that triggers an event such as A/D conversion

F

FIFO	first-in first-out memory buffer—the first data stored is the first data sent to the acceptor. FIFOs are often used on DAQ devices to temporarily store incoming or outgoing data until that data can be retrieved or output. For example, an analog input FIFO stores the results of A/D conversions until the data can be retrieved into system memory, a process that requires the servicing of interrupts and often the programming of the DMA controller. This process can take several milliseconds in some cases. During this time, data accumulates in the FIFO for future retrieval. With a larger FIFO, longer latencies can be tolerated. In the case of analog output, a FIFO permits faster update rates, because the waveform data can be stored on the FIFO ahead of time. This again reduces the effect of latencies associated with getting the data from system memory to the DAQ device.
filtering	a type of signal conditioning that allows you to filter unwanted signals from the signal you are trying to measure
ft	feet

G

GPIB	General Purpose Interface bus, synonymous with HP-IB. The standard bus used for controlling electronic instruments with a computer. Also called IEEE 488 bus because it is defined by ANSI/IEEE Standards 488-1978, 488.1-1987, and 488.2-1987.

H

h	hour
handle	pointer to a pointer to a block of memory; handles reference arrays and strings. An array of strings is a handle to a block of memory containing handles to strings.
handler	a device driver that is installed as part of the operating system of the computer
handshaked digital I/O	a type of digital acquisition/generation where a device or module accepts or transfers data after a digital pulse has been received. Also called latched digital I/O.

hardware	the physical components of a computer system, such as the circuit boards, plug-in boards, chassis, enclosures, peripherals, and cables
hardware triggering	a form of triggering where you set the start time of an acquisition and gather data at a known position in time relative to a trigger signal
hex	hexadecimal
Hz	hertz—the number of scans read or updates written per second

I

IC	integrated circuit
ID	identification
IEEE	Institute of Electrical and Electronics Engineers
IEEE 488	the shortened notation for ANSI/IEEE Standards 488-1978, 488.1-1987, and 488.2-1987. *See also* GPIB.
in.	inches
instrument driver	a set of high-level software functions that controls a specific GPIB, VXI, or RS-232 programmable instrument or a specific plug-in DAQ board. Instrument drivers are available in several forms, ranging from a function callable language to a virtual instrument (VI) in LabVIEW.
interrupt	a computer signal indicating that the CPU should suspend its current task to service a designated activity
interrupt level	the relative priority at which a device can interrupt
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
ISA	industry standard architecture

K

k	kilo—the standard metric prefix for 1,000, or 10^3, used with units of measure such as volts, hertz, and meters
K	kilo—the prefix for 1,024, or 2^{10}, used with B in quantifying data or computer memory
kbytes/s	a unit for data transfer that means 1,000 or 10^3 bytes/s
kS	1,000 samples
Kword	1,024 words of memory

L

LabVIEW	laboratory virtual instrument engineering workbench
LED	light-emitting diode
library	a file containing compiled object modules, each comprised of one of more functions, that can be linked to other object modules that make use of these functions. NIDAQMSC.LIB is a library that contains NI-DAQ functions. The NI-DAQ function set is broken down into object modules so that only the object modules that are relevant to your application are linked in, while those object modules that are not relevant are not linked.

M

m	meters
M	(1) Mega, the standard metric prefix for 1 million or 10^6, when used with units of measure such as volts and hertz; (2) mega, the prefix for 1,048,576, or 2^{20}, when used with B to quantify data or computer memory
MB	megabytes of memory
Mbytes/s	a unit for data transfer that means 1 million or 10^6 bytes/s
memory buffer	*See* buffer.
MIO	multifunction I/O

MIPS	million instructions per second—the unit for expressing the speed of processor machine code instructions
MITE	MXI Interface to Everything—a custom ASIC designed by National Instruments that implements the PCI bus interface. The MITE supports bus mastering for high-speed data transfers over the PCI bus.
MS	million samples
MSB	most significant bit

N

NB	NuBus—a slot-dependent, 32-bit bus type used in Macintosh computers that has 32 interrupts and does not use DMA
NI-DAQ	National Instruments driver software for DAQ hardware
nodes	execution elements of a block diagram consisting of functions, structures, and subVIs

O

onboard channels	channels provided by the plug-in data acquisition board
onboard RAM	optional RAM usually installed into SIMM slots
operating system	base-level software that controls a computer, runs programs, interacts with users, and communicates with installed hardware or peripheral devices

P

parallel mode	a type of SCXI operating mode in which the module sends each of its input channels directly to a separate analog input channel of the device to the module
passband	the range of frequencies which a device can properly propagate or measure
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 work-stations; it offers a theoretical maximum transfer rate of 132 Mbytes/s.

PID control	a three-term control mechanism combining proportional, integral, and derivative control actions. Also see proportional control, integral control, and derivative control.
pipeline	a high-performance processor structure in which the completion of an instruction is broken into its elements so that several elements can be processed simultaneously from different instructions
port	(1) a communications connection on a computer or a remote controller (2) a digital port, consisting of four or eight lines of digital input and/or output
ppm	parts per million
pretriggering	the technique used on a DAQ board to keep a continuous buffer filled with data, so that when the trigger conditions are met, the sample includes the data leading up to the trigger condition
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
pts	points
pulse trains	multiple pulses
pulsed output	a form of counter signal generation by which a pulse is outputted when a counter reaches a certain value

R

RAM	random-access memory
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
resolution	the smallest signal increment that can be detected by a measurement system. Resolution can be expressed in bits, in proportions, or in percent of full scale. For example, a system has 12-bit resolution, one part in 4,096 resolution, and 0.0244% of full scale.
resource locking	a technique whereby a device is signaled not to use its local memory while the memory is in use from the bus

retry	an acknowledge by a destination that signifies that the cycle did not complete and should be repeated
ROM	read-only memory

S

s	seconds
S	samples
sample counter	the clock that counts the output of the channel clock, in other words, the number of samples taken. On boards with simultaneous sampling, this counter counts the output of the scan clock and hence the number of scans.
scan	one or more analog or digital input samples. Typically, the number of input samples in a scan is equal to the number of channels in the input group. For example, one pulse from the scan clock produces one scan which acquires one new sample from every analog input channel in the group.
scan clock	the clock controlling the time interval between scans. On boards with interval scanning support (for example, the AT-MIO-16F-5), this clock gates the channel clock on and off. On boards with simultaneous sampling (for example, the EISA-A2000), this clock clocks the track-and-hold circuitry.
scan rate	the number of scans per second. For example, a scan rate of 10 Hz means sampling each channel 10 times per second.
sensor	a device that responds to a physical stimulus (heat, light, sound, pressure, motion, flow, and so on), and produces a corresponding electrical signal
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.
signal conditioning	the manipulation of signals to prepare them for digitizing
SIMM	single in-line memory module
SMB	a type of miniature coaxial signal connector
soft reboot	restarting a computer without cycling the power, usually via the operating system

software trigger	a programmed event that triggers an event such as data acquisition
software triggering	a method of triggering in which you simulate an analog trigger using software. Also called conditional retrieval.
S/s	samples per second—used to express the rate at which a DAQ board samples an analog signal
STC	system timing controller
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
system RAM	RAM installed on a personal computer and used by the operating system, as contrasted with onboard RAM

T

TC	terminal count—the highest value of a counter
thermocouple	a temperature sensor created by joining two dissimilar metals. The junction produces a small voltage as a function of the temperature.
throughput rate	the data, measured in bytes/s, for a given continuous operation, calculated to include software overhead.
transfer rate	the rate, measured in bytes/s, at which data is moved from source to destination after software initialization and set up operations; the maximum rate at which the hardware can operate
trigger	any event that causes or starts some form of data capture
TTL	transistor-transistor logic

U

update	the output equivalent of a scan. One or more analog or digital output samples. Typically, the number of output samples in an update is equal to the number of channels in the output group. For example, one pulse from the update clock produces one update which sends one new sample to every analog output channel in the group.
update rate	the number of output updates per second

V

V	volts
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
VISA	virtual instrument software architecture—a new driver software architecture developed by National Instruments to unify instrumentation softwareGPIB, DAQ, and VXI. It has been accepted as a standard for VXI by the VXIplug&play Systems Alliance.
VPICD	virtual programmable interrupt controller device

W

waveform	multiple voltage readings taken at a specific sampling rate
wire	data path between nodes
word	the standard number of bits that a processor or memory manipulates at one time. Microprocessors typically use 8-, 16-, or 32-bit words.

Index