Accessing I/O with the NI Scan Engine

June 2008, 372521A–01

Use I/O variables to access coherent sets of I/O channels through the NI Scan Engine.

To view related topics, click the **Locate** button, shown at left, in the toolbar at the top of this window. The *LabVIEW Help* highlights this topic in the **Contents** tab so you can navigate the related topics.

© 2008 National Instruments Corporation. All rights reserved.

Using I/O Variables (NI Scan Engine)

An I/O variable is a type of shared variable that is tied to a physical I/O channel. Use I/O variables to read and write I/O channels from a VI. I/O variables use the NI Scan Engine to enable efficient, non-blocking access to coherent sets of I/O channels. The NI Scan Engine implements a scan that stores data in a global memory map and updates values at a fixed interval, known as the scan period.

Because I/O variables communicate with physical I/O channels, you cannot create new I/O variables. To add a new I/O variable to a project, you must first connect the associated I/O module(s) to the target. When you add a new target with the NI Scan Engine installed to a LabVIEW project, LabVIEW automatically detects the I/O modules attached to the target and creates an I/O variable in the project for each I/O channel. When you add a new I/O module to a project under a target with the NI Scan Engine installed, LabVIEW automatically adds an I/O variable for each channel of the I/O module. If you add or remove I/O modules under a target that is already part of the current LabVIEW project, you can use the Project & System Comparison dialog box to ensure that the project reflects the I/O modules attached to the target.

Note When you run a VI that contains I/O Variables, LabVIEW automatically deploys the corresponding I/O modules. When deploying an I/O module, LabVIEW deploys all the I/O variables within the module as a group.

You can use the **Project Explorer** window to edit the name, description, scaling, and network-publishing of an I/O variable. You also can use the multiple variable editor to edit multiple I/O variables.

I/O variables support timestamps. However, timestamps require additional memory and CPU overhead and can affect the determinism of the application. You should enable timestamps only if you plan to use the timestamps.

Note If the interface software for an I/O bus fails to initialize, LabVIEW aborts loading all I/O variables in the project. In this case, the I/O variables might still appear in the project, but if an application attempts to access the variables, a run-time error occurs.

Creating I/O Aliases

You can create aliases of an I/O variable to provide an extra layer of abstraction from the physical I/O channel. You also can create aliases of other aliases, resulting in a chain of aliases. The value of an I/O alias is linked bidirectionally to the value of the parent, so updating the value of the parent updates the value of the I/O alias, and updating the value of the I/O alias updates the value of the parent. By extension, updating any link in a chain of I/O aliases updates all other links in the chain.

To create an I/O alias variable, right-click an RT target that contains I/O variables in the **Project Explorer** window and select **New»Variable** from the shortcut menu to display the Shared Variable Properties. Enter a name for the I/O alias and select **I/O Alias** from the **Variable Type** pull-down menu. Then click the **Browse...** button and select the existing I/O variable or I/O alias to which you want to bind the new I/O alias.

Note When you bind an alias to an I/O channel, ensure that the data type of the alias matches the data type of the I/O channel.

To batch create I/O aliases, first create a single I/O alias to use as a template for the batch creation process. Right-click a variable library that contains an existing I/O alias and select **Create Variables...** from the shortcut menu to display the Batch Variable Creation dialog box. Select **Copy properties from**, click the **Browse** button, and select the existing I/O alias that you want to use as a template. Enter the number of I/O aliases you want to create in the **Number to create** field and click the **OK** button to create the new I/O alias variables. LabVIEW creates the I/O alias variables and binds them all to the same I/O variable as the template variable. LabVIEW automatically opens the Multiple Variable Editor window, which you can use to edit the new variables. For example, you might want to edit the **Alias Path** of each I/O alias to bind to a unique I/O variable.

You can use the **Project Explorer** window to edit the name, description, scaling, and network-publishing of an I/O alias. You also can use the **Multiple Variable Editor** window to edit multiple I/O aliases.

Note You can enable network publishing on an I/O alias variable, but you cannot deploy a library that contains both network-published I/O alias variables and other types of network-published shared variables. Instead, you must deploy network-published I/O alias variables in a separate library.

Scaling Values

You can enable linear scaling on an I/O variable or alias on the Scaling page of the **Shared Variable Properties** dialog box. For example, if you have an I/O variable connected to a thermocouple input, you could create a Celsius alias and a Fahrenheit alias. Then you could scale each alias and use the aliases to display the temperature in both units of measure.

Note Some I/O busses implement hardware scaling. In this case, LabVIEW I/O variable scaling provides an additional scaling layer. Rather than overwrite the hardware scaling, LabVIEW uses the hardware-scaled value from the I/O bus as the raw value when calculating the software-scaled value.

Forcing Values

Forcing an I/O variable causes the associated I/O channel to assume the value you specify until you unforce the variable, reboot the target, or force the variable to assume a different value. As long as an I/O variable is forced, the scan engine does not update the value of the variable. Unforcing an I/O variable returns control of the I/O value to the scan engine.

Note When an I/O variable is forced, each I/O variable access takes slightly longer than when the variable is not forced, which could cause a loop to run late if the loop period is not long enough to accommodate the forcing overhead.

Use the NI Distributed System Manager to force or unforce the value of an I/O variable during debugging or to manually control an I/O channel. From LabVIEW, select **Tools»Distributed System Manager** to launch the NI Distributed System Manager.

Note You can use the NI Distributed Manager security settings to prevent users from forcing I/O variables from within the NI Distributed System Manager.

Use the Forcing VIs to force and unforce I/O variable values programmatically.

Forcing applies to aliases as well as standard I/O variables. When you force an I/O variable, you also force all associated aliases. When you force an alias, you also force the parent I/O variable and all other associated aliases. LabVIEW applies scaling to forced values as if the forced value were the actual value of the I/O channel. So, when you force one link in a chain of aliases, all links in the chain scale appropriately.

You can enable and disable global forcing on a target in System Manager by using the **Enable Forcing** and **Disable Forcing** buttons. You also can enable and disable global forcing on a target programmatically using the Enable Variable Forcing and Disable Variable Forcing VIs. You can use the global forcing state of the target for batch forcing of I/O variables. To perform batch I/O variable forcing, disable forcing on the target, set the desired forcing values for all I/O variables, then re-enable forcing on the target.

Note Forcing does not persist if the target reboots.

Network-Publishing I/O Variables

Use the Shared Variable Properties dialog box to enable and disable network publishing on an I/O variable. Use network-publishing if you need to monitor I/O values on a host computer or access an I/O variable from a remote target. Use the Scan Engine page to set the global network-publishing rate for all I/O variables on a target.

Note Accessing I/O variables remotely is not deterministic. If you do not plan to access I/O variables remotely, disable network publishing to minimize overhead. When you disable network publishing on an I/O variable or I/O alias, you cannot access the variable from a VI running on another machine. However, you can still force the variable from the NI Distributed System Manager. To prevent users from forcing an I/O variable, use the System Manager security settings.

Configuring I/O Variable Properties at Run Time

You can update description, network, and scaling options of an I/O variable at run time. However, you cannot update the name or data type of an I/O variable at run time.

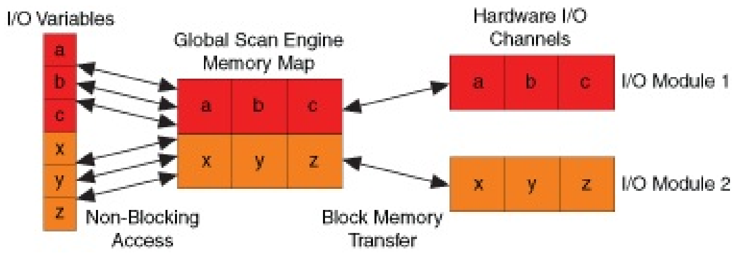
I/O Variable Access Methods

LabVIEW adds I/O variables to a global scan engine memory map and updates the values of all I/O variables concurrently. However, you can configure each I/O variable node to use either scanned access or direct access. To switch from scanned to direct access, right-click an I/O variable node on the block diagram and select **Change to Direct**. To switch from direct to scanned access, right-click an I/O variable node on the block diagram and select **Change to Scanned**.

Note LabVIEW adds all I/O variables to the global scan engine memory map, regardless of the access mode you use for each variable.

Scanned I/O Access

By default, LabVIEW configures I/O variable nodes to use scanned I/O access. Use scanned access for coherent sets of I/O channels that update at a single rate and for expansion I/O channels. Scanned I/O access uses the scan engine memory map to perform non-blocking I/O reads and writes, as shown in the following illustration.

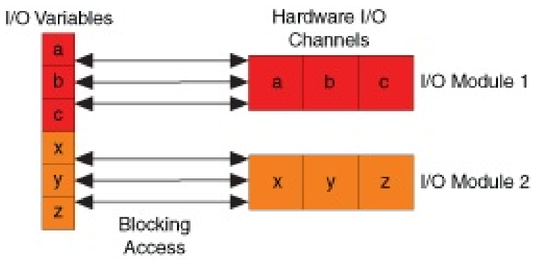


Each time you write to an I/O variable using scanned access, you overwrite the previous value stored in the scan engine memory map. During each scan, LabVIEW pushes the value stored in the memory map to the physical I/O channel. To prevent data loss, you must synchronize all I/O variable write operations to the scan period.

Each time you read from an I/O variable using scanned access, the scan engine returns the most recent value stored in the memory map. During each scan, LabVIEW reads the most recent value from the physical I/O channel and writes that value to the memory map.

Direct I/O Access

Use direct I/O access for single-point local I/O channels with rates asynchronous to the scan period. Direct I/O access bypasses the scan engine memory map and communicates directly with the I/O device driver to perform blocking I/O reads and writes, as shown in the following illustration.



Note The speed of direct I/O access varies by controller, I/O module, and communication protocol. Refer to the specific hardware documentation for more information about I/O access speed.

Choosing an I/O Access Method

In general, scanned I/O access is appropriate for groups of I/O channels with similar update rates, and direct I/O access is appropriate for individual I/O channels that update faster than the scan period. The following table summarizes when to use each I/O access method:

Access Method	Suggested Use
Scanned	Expansion I/O; coherent sets of single-point, single-rate I/O channels
Direct	Individual single-point local I/O channels asynchronous to the scan period

Using the NI Scan Engine

The NI Scan Engine is a software component that you can install on supported targets, such as RT Series PXI and CompactRIO targets. Refer to the target I/O driver documentation for information about NI Scan Engine support.

Note You must install the LabVIEW Real-Time Module to create applications that use the NI Scan Engine. However, if you do not have the Real-Time Module installed on your computer, you can still use the NI Distributed System Manager to monitor and manage scan engine settings on targets with the NI Scan Engine installed.

The NI Scan Engine enables efficient access to coherent sets of data channels, such as I/O channels, using a scan that stores data in a global memory map and updates all values at a single rate, known as the scan period.

Note By default, the NI Scan Engine runs in a thread above time-critical priority, although LabVIEW includes the scan thread in the time-critical category when reporting CPU usage statistics. If you plan to use the scan engine, you must synchronize the deterministic sections of the application with the scan period to ensure that the scan thread does not affect the determinism of the application. If you do not plan to use I/O variables on a target, do not install the NI Scan Engine on the target. If the NI Scan Engine is already installed on the target, you can use the NI Measurement & Automation Explorer (MAX) to uninstall the NI Scan Engine.

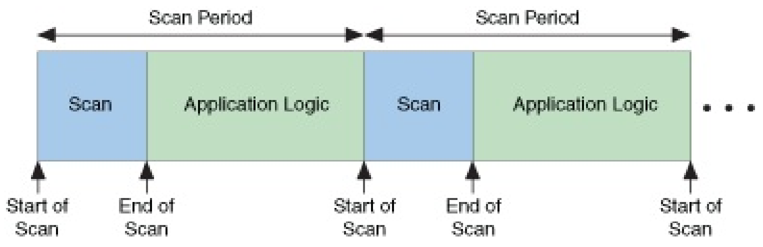
Configuring Scan Engine Settings

Use the Scan Engine page to configure scan engine settings including the scan period, network-publishing rate, and priority level of the NI Scan Engine.

Use the NI Distributed System Manager to monitor and manage scan engine faults and modes. You also can use the NI Scan Engine VIs to view and configure scan engine settings programmatically.

Scan Engine Timing

The NI Scan Engine executes at regular intervals determined by the **Scan Period** you specify on the **Scan Engine** page. Choose a period long enough to accommodate both the scan itself and the application logic, as shown in the following illustration.



Note The length of the scan depends on the number and type of I/O items deployed to the target. To maximize scan engine performance, undeploy any I/O items that you do not plan to use in the application.

Use the Get Scan Engine Period VI to read the scan period programmatically. Use the Set Scan Engine Period VI to set the scan period programmatically.

Note Real-time loops generally need one or two warm-up iterations to begin executing deterministically. Before checking to ensure that an application meets timing requirements, you should allow each time-critical loop to execute warm-up iterations.

Synchronizing to the Scan Engine

By default, the scan engine runs in a thread above time-critical priority. You should synchronize time-critical code to the scan engine to avoid collisions that could affect the determinism of the application.

Use the **Synchronize to Scan Engine** timing source to synchronize timed structure execution to the scan engine. If you do not want to use a timed structure, you can use the Synchronize to Scan Engine VI to synchronize to the scan engine. Both synchronization methods trigger execution at the time labeled **End of Scan** in the previous illustration. To use I/O variables as a coherent data set, you should ensure that the synchronized code finishes executing before the next scan iteration. However, you can safely skip scan iterations if the code does not depend on a coherent data set.

Note If synchronized code does not finish executing before the next scan iteration, the information reported by the error cluster of an I/O variable might lose synchronization with the I/O value.

Note When you press the **Abort** button on a VI that involves synchronization to the NI Scan Engine, the VI does not abort until the current scan iteration completes, so the VI could appear to hang temporarily if the scan period is sufficiently long.

Setting the Priority of the NI Scan Engine

By default, the NI Scan Engine runs in a thread above time-critical priority. For most applications that use the scan engine, above time-critical is the appropriate priority because reading and writing I/O values is generally the highest-priority task of an application. To prevent jitter, you must synchronize time-critical code, such as control loops, to the scan engine and configure the scan period to accommodate the time-critical code.

For applications in which I/O is not the highest-priority task, you also can configure the priority of the scan engine to fall between time-critical and Timed Structure priority.

NI Scan Engine Modes

Note Only certain targets and devices use NI Scan Engine modes. CompactRIO targets with local I/O modules do not use NI Scan Engine modes. Refer to the I/O hardware documentation for information about hardware-specific mode behavior.

LabVIEW distinguishes four NI Scan Engine modes:

Initialization Mode	Occurs only briefly during startup.
Configuration Mode	The required mode when configuring scan engine settings on hardware that uses NI Scan Engine modes.
Active Mode	The mode in which the scan engine runs and updates values.
Fault Mode	The mode triggered when a major or unrecoverable fault occurs.

Note Fault mode behavior varies by target. Refer to the specific target hardware documentation for information about fault mode behavior.

Use the NI Distributed System Manager to view and configure the scan engine mode. Use the Get Scan Engine Mode VI to read the scan engine mode programmatically. Use the Set Scan Engine Mode VI to set the scan engine mode programmatically.

NI Scan Engine Faults

Targets with the NI Scan Engine installed use faults to address asynchronous error conditions. LabVIEW distinguishes three fault levels: minor, major, and unrecoverable. LabVIEW logs all faults in the NI Distributed System Manager.

Note Some I/O hardware drivers implement fault handling behavior in response to major faults. Refer to the specific hardware documentation for information about fault handling behavior.

Examples of minor faults include startup errors, which can occur when the controller is unable to apply its saved configuration on startup. If LabVIEW detects that the scan engine has run late, LabVIEW triggers major fault −66460. If LabVIEW detects ten consecutive late scan iterations, LabVIEW triggers major fault −66461 and the NI Scan Engine stops running. Unrecoverable faults can occur due to a hardware failure or software crash. In the event of an unrecoverable fault, reboot the controller and contact National Instruments.

Viewing and Clearing Faults

Use the NI Distributed System Manager to view and clear faults. You also can use the Get Fault List VI and the Clear Fault VI to view and clear faults programmatically.

System Faults

LabVIEW defines a set of common faults and you can log additional faults based on LabVIEW error clusters. Fault codes are grouped together by type, as shown in the following table.

Fault Type	**Range of Fault Codes**
I/O Scan Driver Errors	−66000 through −66099
I/O Variables	−66200 through −66299
NI Scan Engine VIs	−66300 through −66399
NI Scan Engine	−66400 through −66499
User Fault	Any LabVIEW error code

User Faults

You can use the Set Fault VI to trigger minor or major user-defined faults based on LabVIEW error codes. LabVIEW reserves a memory pool large enough to log up to 100 unique fault codes. If you exceed 100 unique fault codes, LabVIEW triggers minor fault −66420 and discontinues logging additional faults. However, even if the maximum number of user faults has been reached, a major or unrecoverable fault can still trigger hardware drivers to initiate fault handling behavior.

Note Refer to the specific hardware documentation for information about fault handling behavior.