LabVIEW™ Upgrade Notes

Version 8.0

These upgrade notes describe the process of upgrading LabVIEW for Windows, Mac OS, and Linux to version 8.0, issues you might encounter when you upgrade, and new features.

Refer to the *LabVIEW Help* for more information about LabVIEW 8.0 features, as well as for information about LabVIEW programming concepts, step-by-step instructions for using LabVIEW, and reference information about LabVIEW VIs, functions, palettes, menus, and tools. The *LabVIEW Help* also lists the LabVIEW documentation resources available from National Instruments. Access the *LabVIEW Help* by selecting **Help»Search the LabVIEW Help**.

Contents

NATIONAL INSTRUMENTS™

Upgrading to LabVIEW 8.0

If you are upgrading from a previous version of LabVIEW, refer to the *Converting VIs*, the *Upgrading Toolkits, Instrument Drivers, and Add-Ons*, and the *Upgrade and Compatibility Issues* sections of this document first.

Converting VIs

When you open a VI last saved in LabVIEW 4.0 or later, LabVIEW 8.0 automatically converts and compiles the VI. You must save the VI in LabVIEW 8.0, or the conversion process, which uses extra memory resources, occurs every time you access the VI.

Also, you might experience a large run-time degradation of performance for any VI that has unsaved changes, including a recompile. Refer to the *LabVIEW Help* for more information about this performance and memory issue.

Note VIs you save in LabVIEW 8.0 do not load in earlier versions of LabVIEW. Select **File»Save for Previous Version** to save VIs so they can run in LabVIEW 7.1. Before saving VIs in LabVIEW 8.0, keep a backup copy of VIs you plan to use in LabVIEW 7.1 or earlier.

If your computer does not have enough memory to convert all the VIs at once, convert the VIs in stages. Examine the hierarchy of VIs you want to convert and begin by loading and saving subVIs in the lower levels of the hierarchy. Then progress gradually to the higher levels of the hierarchy. Open and convert the top-level VI last. You also can select **Tools»Advanced»Mass Compile** to convert a directory of VIs. However, mass compiling converts VIs in a directory or LLB in alphabetical order. If the conversion process encounters a high-level VI first, mass compiling requires approximately the same amount of memory as if you opened the high-level VI first.

You can monitor memory usage by selecting **Help»About LabVIEW** to display a summary of the amount of memory you currently are using.

Upgrading Toolkits, Instrument Drivers, and Add-Ons

After you install LabVIEW 8.0, make sure you have a compatible version of any toolkits and add-ons, and reinstall the toolkits and add-ons in the LabVIEW 8.0 directory. You first might need to uninstall the toolkit from previous versions of LabVIEW. Refer to the documentation for the LabVIEW toolkit or add-on for more information about installation.

You also must mass compile existing toolkit, instrument driver, and add-on VIs for use in LabVIEW 8.0. Refer to the *Converting VIs* section of this document for more information about mass compiling VIs.

The following toolkits, instrument drivers, and add-ons require upgrades or downloads for use in LabVIEW 8.0.

- If you have the LabVIEW Application Builder, you must upgrade to LabVIEW Application Builder 8.0. The LabVIEW 8.0 Professional

Development System includes Application Builder 8.0. Refer to the *LabVIEW Application Builder Readme* located in the `labview\readme` directory for more information about installing the LabVIEW Application Builder.

- You must use VI Analyzer 1.1 in LabVIEW 8.0. Refer to the National Instruments Web site at `ni.com/info` and enter the info code `exd8yy` to access the Upgrade Advisor and download VI Analyzer 1.1.
- You must download additional VIs to use the Internet Toolkit 6.0 with LabVIEW 8.0. Refer to the National Instruments Web site at `ni.com/info` and enter the info code `exkrkb` to download the necessary VIs.
- The instrument driver for the HP/Agilent 34401A Digital Multimeter (DMM) now more closely resembles the National Instruments DMM template driver. This driver is not compatible with the HP34401A driver that LabVIEW 7.*x* and earlier use. If you need compatibility with the LabVIEW 7.*x* HP34401A driver, download that driver from the National Instruments Instrument Driver Network at `ni.com/idnet`.

Upgrading Additional National Instruments Software

You must use NI TestStand 3.5 or later in LabVIEW 8.0. Refer to the National Instruments Web site at `ni.com/info` and enter the info code `exd8yy` to access the Upgrade Advisor and download NI TestStand 3.5 or later.

You must use Logos 4.6 or later in LabVIEW 8.0. Refer to the National Instruments Web site at `ni.com/info` and enter the info code `exymjt` to download the latest version of Logos.

Upgrading from Previous Versions of LabVIEW

Upgrading to new versions of LabVIEW does not affect previous versions of LabVIEW on the computer because the new versions install in a different directory. LabVIEW 5.*x* and earlier install in the `labview` directory. LabVIEW 6.0 and later install in the `labview` *x.x* directory, where *x.x* is the version number. You can install LabVIEW 8.0 without uninstalling previous versions of LabVIEW.

To replace your existing version of LabVIEW, uninstall the existing version of LabVIEW, run the LabVIEW 8.0 installer, and set the default installation directory to the same `labview` directory where you installed the previous version of LabVIEW.

(Windows) You also can replace the existing version of LabVIEW with LabVIEW 8.0 by using the Add/Remove Programs applet in the Control

Panel to uninstall the existing version of LabVIEW. The uninstaller does not remove any files you created in the `labview` directory.

Note When you uninstall or reinstall LabVIEW, LabVIEW uninstalls the `.llb` files in the `vi.lib` directory, including any VIs and controls you saved in the `.llb` files. Save your VIs and controls in the `user.lib` directory to add them to the **Controls** and **Functions** palettes.

To use LabVIEW environment settings from a previous version of LabVIEW, copy the LabVIEW preferences file from the `labview` directory in which the previous version is installed.

Caution If you replace the LabVIEW 8.0 preferences file with a preferences file from a previous version, you might override preference settings added to LabVIEW since the previous version.

After you install LabVIEW 8.0, copy the LabVIEW preferences file to the LabVIEW 8.0 directory.

(Windows) LabVIEW stores preferences in the `labview.ini` file.

(Mac OS) LabVIEW stores preferences in the `LabVIEW Preferences` file in the `Library:Preferences` folder in your home directory.

(Linux) LabVIEW stores preferences in the `.labviewrc` file in your home directory.

To use files from the `user.lib` directory of a previous version of LabVIEW, copy the files from the `labview` directory in which the previous version is installed. After you install LabVIEW 8.0, copy the files to the `user.lib` directory in the LabVIEW 8.0 directory.

Upgrade and Compatibility Issues

Refer to the following sections for upgrade and compatibility issues specific to different versions of LabVIEW.

Upgrading from LabVIEW 7.*x*

You might encounter the following compatibility issues when you upgrade to LabVIEW 8.0 from LabVIEW 7.*x*.

Platforms Supported

LabVIEW 8.0 includes the following changes in platforms supported:

- LabVIEW 7.1 and later do not support Windows Me/98/95. LabVIEW 8.0 does not support Windows NT.
- LabVIEW 8.0 supports Windows XP x64, but current NI driver support for XP x64 is limited.
- LabVIEW 8.0 does not support Mac OS X 10.2 or earlier.
- LabVIEW 8.0 does not support Sun Solaris.

System Requirements

LabVIEW 7.*x* requires a minimum of 128 MB of RAM, but National Instruments recommends 256 MB of RAM. LabVIEW 8.0 requires a minimum of 256 MB of RAM, but National Instruments recommends 512 MB of RAM.

LabVIEW 7.*x* requires a screen resolution of 800 × 600 pixels, but National Instruments recommends a screen resolution of 1,024 × 768 pixels. LabVIEW 8.0 requires a screen resolution of 1,024 × 768 pixels.

(Windows) LabVIEW 7.*x* requires a minimum of a Pentium III or greater or Celeron 600 MHz or equivalent processor, but National Instruments recommends a Pentium 4 or equivalent processor. LabVIEW 8.0 requires a minimum of a Pentium III or Celeron 866 MHz or equivalent processor, but National Instruments recommends a Pentium 4/M or equivalent processor.

(Linux) LabVIEW 7.*x* requires a minimum of a Pentium III or greater or Celeron 600 MHz or equivalent processor, but National Instruments recommends a Pentium 4 or equivalent processor. LabVIEW 8.0 requires a minimum of a Pentium III or Celeron 866 MHz or equivalent processor, but National Instruments recommends a Pentium 4/M or equivalent processor.

(Windows) LabVIEW 7.*x* requires at least 130 MB of disk space for the minimum LabVIEW installation or 550 MB disk space for the complete LabVIEW installation. LabVIEW 8.0 requires at least 900 MB of disk space for the minimum LabVIEW installation or 1.2 GB disk space for the complete LabVIEW installation.

(Mac OS) LabVIEW 7.*x* requires at least 280 MB of disk space for the minimum LabVIEW installation or 350 MB disk space for the complete LabVIEW installation. LabVIEW 8.0 requires at least 500 MB of disk space for the minimum LabVIEW installation or 600 MB disk space for the complete LabVIEW installation.

(Linux) LabVIEW 7.*x* requires at least 200 MB of disk space for the minimum LabVIEW installation or 300 MB disk space for the complete

LabVIEW installation. LabVIEW 8.0 requires at least 400 MB of disk space for the minimum LabVIEW installation or 550 MB disk space for the complete LabVIEW installation.

(Linux) LabVIEW 7.*x* requires GNU C Library (`glibc`) version 2.1.3 or later, but National Instruments recommends GNU C Library version 2.2.4 or later. LabVIEW 8.0 requires GNU C Library version 2.2.4 or later.

(Linux) LabVIEW 7.*x* runs on Red Hat Linux 7.0 or later, Mandrake Linux 8.0 or later, SuSE Linux 7.1 or later, or Debian Linux 3.0 or later. LabVIEW 8.0 runs on Red Hat Enterprise Linux WS 3 or later, MandrakeLinux/Mandriva 10.0 or later, or SuSE Linux 9.1 or later.

Custom Palette Views

LabVIEW 8.0 does not support custom palette views. You can edit a palette set without using a custom palette view. Refer to the *Palette Editing Enhancements* section of this document for more information about palette changes in LabVIEW 8.0.

VI and Function Behavior Changes

The behavior of the following VIs and functions changed in LabVIEW 8.0.

.NET VIs and Applications

You must have the .NET Framework 1.1 Service Pack 1 or later to use .NET functions and applications in LabVIEW 8.0. You must remove Microsoft .NET Framework 1.1 Hotfix KB886904 before installing the .NET Framework 1.1 Service Pack 1.

If you load a .NET VI last saved in LabVIEW 7.*x*, LabVIEW 8.0 might prompt you to find the assemblies to which that VI refers even if the assembly files are in the same directory as the VI or if you registered them by selecting **Tools»Advanced».NET Assembly References** in LabVIEW 7.*x*.

Analyze VI Algorithms

In LabVIEW 7.1 and later, the Analyze VIs use the BLAS/LAPACK algorithms. These VIs now produce more accurate results. In LabVIEW 8.0, these VIs are on the **Mathematics** and **Signal Processing** palettes.

Append Signals Express VI

In LabVIEW 7.*x*, if **Input Signal A** of the Append Signals Express VI is empty or not wired and you wire a single signal or a combined signal to **Input Signal B**, the **Appended Signals** output is empty. In LabVIEW 8.0, if **Input Signal A** is empty or not wired and you wire a single signal to **Input Signal B**, the Express VI returns **Input Signal B**. If you wire only a combined signal to **Input Signal B**, each signal in the combined signal appends the following signal to create one signal as a result.

Comparison Functions

In LabVIEW 7.*x* and earlier, when you use the Comparison functions to compare variant data, LabVIEW first compares the length of the two variants and then compares the variants bit by bit. LabVIEW 8.0 begins the comparison of variant data with the type codes, which encode the actual type information of the variants, and then compares other type-specific attributes.

Dot Product VI

In LabVIEW 7.0, the Dot Product VI calculates the dot product of input vectors *X* and *Y* using the following equation:

$$X^*Y = \sum_{i=0}^{n-1} x_i y_i$$

In LabVIEW 7.1 and later, the Dot Product VI calculates the dot product of complex inputs using the following equation:

$$X^*Y = \sum_{i=0}^{n-1} x_i y_i^*$$

where y_i^* is the complex conjugate of y_i.

Easy Text Report VI (Mac OS and Linux)

The connector pane of the Easy Text Report VI changed. In LabVIEW 8.0, when you open a VI last saved in LabVIEW 7.*x* or earlier that uses the Easy Text Report VI, you must right-click the subVI and select **Relink To SubVI** from the shortcut menu.

Format Into String Function

In LabVIEW 7.*x*, using the `%o` or `%x` format specifier syntax elements with the Format Into String function rounds a floating-point input to a 32-bit integer before converting that input to a string.

In LabVIEW 8.0, these format specifier syntax elements cause this function to round floating-point inputs to 64-bit integers before converting the inputs to strings.

Join Numbers Function

In LabVIEW 7.*x* and earlier, the Join Numbers function coerces 32-bit integer inputs to 16-bit integers to create one 32-bit integer. In LabVIEW 8.0, the Join Numbers function joins 32-bit integer inputs to create one 64-bit integer.

Note If you open a LabVIEW 7.*x* VI in LabVIEW 8.0, LabVIEW coerces 32-bit integer inputs to 16-bit integers.

Number to String Conversion Functions

In LabVIEW 7.*x*, the Number to Hexadecimal String, Number to Octal String, and Number to Decimal String functions round a floating-point input to a 32-bit integer before converting that input to a string.

In LabVIEW 8.0, these functions round floating-point inputs to 64-bit integers before converting the inputs to strings. However, if you open a LabVIEW 7.*x* VI in LabVIEW 8.0, LabVIEW maintains compatibility and functionality by rounding floating-point inputs to 32-bit integers.

Open VI Reference Function

In LabVIEW 7.*x*, if the **vi path** input of the Open VI Reference function is a path and a VI in memory exists with the same name, LabVIEW returns a reference to the VI in memory, even if the path to the VI in memory does not match the path you specified.

In LabVIEW 8.0, if the **vi path** input of the Open VI Reference function is a string, LabVIEW opens the VI only if **vi path** matches the qualified filename of a VI in memory on that target. If **vi path** is a path, LabVIEW searches for a VI in memory with the same path on the same target. If LabVIEW does not find a VI with a matching path, LabVIEW tries to load the VI from disk at the specified path. An error occurs if LabVIEW cannot find the file or if the file conflicts with another VI in memory. Refer to the *LabVIEW Project Libraries* section of this document for more information about qualified filenames. Refer to the *LabVIEW Projects* section of this document for more information about targets.

Quick Scale VI

In LabVIEW 7.1 and earlier, if the **X** input of the Quick Scale 1D VI or the Quick Scale 2D VI is an array of zeros, this VI returns **max|X|** as `0` and **Y[i]=X[i]/Max|X|** or **Yij=Xij/Max|X|** as an array of `NaN`. In LabVIEW 8.0, if the **X** input of the Quick Scale VI is an array of zeros, this VI returns **max|X|** as `0` and **Y[i]=X[i]/Max|X|** or **Yij=Xij/Max|X|** as an array of zeros.

Read Key VI

In LabVIEW 7.*x* and earlier, you can use the Read Key VI to read a Japanese multibyte-character string encoded in Shift-JIS. You must wire `1` or `<Shift-JIS>` to the **multibyte encoding** input. In LabVIEW 8.0, the Read Key VI reads multibyte-character, encoded strings by default if you set the operating system locale to the appropriate encoding.

Scale VI

In LabVIEW 7.1 and earlier, if the **X** input of the Scale 1D VI or the Scale 2D VI is an array of zeros, this VI returns **scale** as `0`, **offset** as `0`, and **Y=(X–offset)/scale** as an array of `NaN`. In LabVIEW 8.0, if the **X** input of the Scale VI is an array of zeros, this VI returns **scale** as `1`, **offset** as `0`, and **Y=(X–offset)/scale** as an array of zeros.

Semaphore VIs

In LabVIEW 7.*x*, the Release Semaphore VI and the Acquire Semaphore VI do not attempt to execute when the **error in** parameter contains an error. In LabVIEW 8.0, these VIs attempt to execute even if the **error in** parameter contains an error. However, if you open a LabVIEW 7.*x* VI in LabVIEW 8.0, LabVIEW maintains the LabVIEW 7.*x* functionality.

SMTP Email VIs

In LabVIEW 7.*x* and earlier, you can specify a character set by wiring a value to the **character set** input of the SMTP Email VIs. In LabVIEW 8.0, the SMTP Email VIs assume the message is in the system character set. These VIs encode the message into UTF-8 format before sending the email. The SMTP Email VIs no longer have the **character set** or **translit** parameters.

Sort Complex Numbers VI

In LabVIEW 7.*x* and earlier, if you set the **method** input of the Sort Complex Numbers VI to **Magnitude**, LabVIEW does not change the sequence of elements with the same magnitude. In LabVIEW 8.0, if you set **method** to **Magnitude**, LabVIEW sorts elements of the same magnitude first with respect to their real parts and then with respect to their imaginary parts.

Unit Vector VI

In LabVIEW 7.*x* and earlier, the Unit Vector VI calculates the norm of an input vector using the following equation:

$$\|X\| = \sqrt{x_0^2 + x_1^2 + \ldots + x_{n-1}^2}$$

In LabVIEW 8.0, the Unit Vector VI calculates the norm of an input vector using the following equation:

$$\|X\| = \left| |x_0|^y + |x_1|^y + \ldots + |x_{n-1}|^y \right|^{\frac{1}{y}}$$

where X is the input vector, $\|X\|$ is the norm, and y is the norm type.

User VIs

VIs that you place in the `labview\help`, `labview\project`, or `labview\wizard` directories appear in the **Help**, **Tools**, and **File** menus, respectively. VIs that you place in these directories in LabVIEW 7.*x* and earlier might not work as expected in LabVIEW 8.0 because LabVIEW 8.0 opens these VIs in a private application instance.

Use the VIMemory Get VIs in Memory VI in the `labview\vi.lib\Utility\allVIsInMemory.llb` to generate a list of all user VIs in memory in all application instances. Use the Get User Application Reference VI in the `labview\vi.lib\Utility\allVIsInMemory.llb` to create a reference to the current application instance. Refer to the *Working with Application Instances* section of this document for more information about application instances.

Deprecated VIs and Functions

LabVIEW 8.0 does not support the following VIs and functions:

- LabVIEW 7.1 and later do not install the Polynomial Real Zero Counter VI. Use the Polynomial Real Zeros Counter VI instead.
- LabVIEW 7.1 and later do not install the PPC VIs. Use the TCP VIs instead.
- LabVIEW 8.0 does not support the QR Factorization VI. Use the QR Decomposition VI instead.
- LabVIEW 8.0 does not support the Levenberg Marquardt or the Nonlinear Lev-Mar Fit VIs. Use the Nonlinear Curve Fit VI instead.
- In LabVIEW 8.0, the VISA Status Description function is not on the **Functions** palette. Use the Simple Error Handler or General Error Handler VIs instead.

- LabVIEW 8.0 does not support the Chi Square Distribution, F Distribution, Normal Distribution, and T Distribution VIs. Use the Chi-Squared, F, Normal, and Student t instances, respectively, of the Continuous CDF VI instead.
- LabVIEW 8.0 does not support the Inv Chi Square Distribution, Inv F Distribution, Inv Normal Distribution, and Inv T Distribution VIs. Use the Chi-Squared, F, Normal, and Student t instances, respectively, of the Continuous Inverse CDF VI instead.
- In LabVIEW 8.0, the 1D Linear Evaluation VI and the 2D Linear Evaluation VI are not on the **Functions** palette. Use the Linear Evaluation VI instead.
- In LabVIEW 8.0, the 1D Polynomial Evaluation VI and the 2D Polynomial Evaluation VI are not on the **Functions** palette. Use the Polynomial Evaluation VI instead.
- In LabVIEW 8.0, the 1D Rectangular to Polar VI and the 1D Polar to Rectangular VI are not on the **Functions** palette. Use the Re/Im To Polar function and the Polar To Re/Im function instead.
- In LabVIEW 8.0, the Harmonic Analyzer VI is not on the **Functions** palette. Use the Harmonic Distortion Analyzer VI instead to measure the **THD** or **component levels** outputs, or use the SINAD Analyzer VI to measure the **SINAD** or **THD Plus Noise** outputs.
- In LabVIEW 8.0, the Network Functions (avg) VI is not on the **Functions** palette. Use the Frequency Response Function (Mag-Phase), Frequency Response Function (Real-Im), Cross Spectrum (Mag-Phase), or Cross Spectrum (Real-Im) VIs instead.
- In LabVIEW 8.0, the Pulse Parameters VI is not on the **Functions** palette. Use the Transition Measurements VI instead to measure the **slew rate**, **duration**, **overshoot**, or **preshoot** outputs, the Pulse Measurements VI to measure the **period**, **pulse duration**, or **duty cycle** outputs, or the Amplitude and Levels VI to measure the **amplitude**, **high state level**, or **low state level** outputs.
- In LabVIEW 8.0, the Transfer Function VI is not on the **Functions** palette. Use the Frequency Response Function (Mag-Phase) or Frequency Response Function (Real-Im) VIs instead.
- In LabVIEW 8.0, the NI DIAdem Report Wizard Express VI is not on the **Functions** palette. Use the NI DIAdem Report Express VI instead. Refer to the *NI DIAdem Report Express VI* section of this document for more information about the NI DIAdem Report Express VI.

Note You must have DIAdem 9.1 Service Pack 2 or later installed to use the NI DIAdem Report Express VI. Refer to the National Instruments Web site at `ni.com` and enter the info code `exf5ty` to download an evaluation version of DIAdem or purchase DIAdem.

- In LabVIEW 8.0, the VISA Resource Name constant and the IVI Logical Name constant are not on the **Functions** palette. To specify a VISA resource name, use the **VISA resource name** input of the VISA VIs. To specify an IVI logical name, use the appropriate input of the appropriate driver VI that initializes the instrument.
- In LabVIEW 8.0, the error ring constant is not on the **Functions** palette. Use a 32-bit signed integer constant instead to enter the error code that you want.
- **(Windows)** In LabVIEW 8.0, the Sound VIs available on the **Sound** palette in LabVIEW 7.*x* are not on the **Functions** palette. Use the Sound VIs in LabVIEW 8.0 instead. The examples shipped with LabVIEW 7.*x* do not ship with LabVIEW 8.0.
- **(Mac OS and Linux)** The Sound VIs in LabVIEW 8.0 continue to ship with the Sound VIs shipped with LabVIEW 7.1. The examples shipped with LabVIEW 7.*x* do not ship with LabVIEW 8.0.

File I/O VIs and Functions

In LabVIEW 8.0, the Read Characters From File VI is not on the **Functions** palette. Use the Read from Text File function instead.

In LabVIEW 8.0, the Open/Create/Replace File VI is not on the **Functions** palette. Use the Open/Create/Replace File function instead. The following functions include some of the functionality of the Open/Create/Replace File VI in LabVIEW 7.*x* and earlier.

- Use the Get File Size function to determine the size of a file.
- Use the File Dialog Express VI to specify the start path, file pattern, and default name of a file or directory for a dialog box that lets the user browse to a file.
- Use the Refnum to Path function to convert a reference to a path.
- Use the Write to Binary File function to create platform-independent text files or other types of binary files, and use the Read from Binary File function to read the resulting binary files.

In LabVIEW 8.0, the Read File and Write File functions are not on the **Functions** palette. Use the Read from Binary File and Write to Binary File functions instead.

In LabVIEW 8.0, the Write Characters To File VI is not on the **Functions** palette. Use the Write to Text File function instead.

In LabVIEW 8.0, the Access Rights function is not on the **Functions** palette. Use the Get Permissions and Set Permissions functions instead.

In LabVIEW 8.0, the EOF function is not on the **Functions** palette. Use the Get File Size and Set File Size functions instead.

In LabVIEW 8.0, the List Directory function is not on the **Functions** palette. Use the List Folder function instead.

In LabVIEW 8.0, the Lock Range function is not on the **Functions** palette. Use the Deny Access function instead.

If you open a VI built in LabVIEW 7.*x* that includes the New Directory function on the block diagram, LabVIEW 8.0 replaces that function with the Create Folder function. If the folder you specified in the **path** input does not exist, the Create Folder function creates the directory rather than returning an error, as the New Directory function did.

In LabVIEW 8.0, the Seek function is not on the **Functions** palette. Use the Get File Position and Set File Position functions instead.

In LabVIEW 8.0, the Type and Creator function is not on the **Functions** palette. Use the Get Type and Creator and Set Type and Creator functions instead.

In LabVIEW 8.0, the Volume Info function is not on the **Functions** palette. Use the Get Volume Info function instead.

In LabVIEW 8.0, the Open File and New File functions are not on the **Functions** palette. The Read Lines From File VI is not on the **Functions** palette but ships with LabVIEW for compatibility.

In LabVIEW 8.0, the Read From I16 File, Read From SGL File, Write To I16 File, and Write To SGL File VIs are not on the **Functions** palette. Use the Read from Binary File and Write to Binary File VIs instead.

Finding Deprecated VIs

Complete the following steps to find deprecated VIs in the VIs you created in earlier versions of LabVIEW.

1. Load the VIs you want to search into memory.
2. Select **Edit»Find and Replace** to display the **Find** dialog box.
3. Click the **Text** button.
4. From the **Application instance** pull-down menu, select the application instance that contains the VIs you want to search. Refer to the *Working with Application Instances* section of this document for more information about application instances.
5. From the **Search Scope** pull-down menu, select **All VIs in Application Instance**.

6. Place a checkmark in the **Hierarchy window** checkbox.
7. Click the **Find** button.

Property, Method, and Event Behavior Changes

The behavior of the following properties, methods, and events changed in LabVIEW 8.0.

Application Properties and Methods

In LabVIEW 8.0, the behavior of some Application properties and methods depends on the application instance to which they belong. For example, the behavior of the Application:All VIs in Memory property depends on the application instance in which you use it. This property returns a list of all VIs in memory in the same application instance as the property. However, the behavior of the Application:Directory Path property does not depend on the application instance in which you use it. This property returns the absolute path to the directory in which the application is located. This information does not change with each application instance.

Refer to the *Working with Application Instances* section of this document for more information about application instances.

Front Panel:Open Method

The LabVIEW 7.0 Open FP method was renamed to Old Open FP in LabVIEW 7.1. LabVIEW 7.1 includes a different Open FP method that does not return an error if the front panel is already open. The LabVIEW 7.1 Open FP method was renamed to Front Panel:Open in LabVIEW 8.0. If you have VIs that use the Old Open FP method from LabVIEW 7.0, replace the method with the Front Panel:Open method.

Key Down and Key Repeat Events

The **VKey** data field of the Key Down, Key Down?, Key Repeat, and Key Repeat? events for VIs and controls now has separate values for the <Return> key on the alphanumeric section of the keyboard and the <Enter> key on the numeric keypad. In LabVIEW 7.*x* and earlier, when the <Enter> key or the <Return> key generates one of these events, LabVIEW returns **`<Enter>`** in the **VKey** data field. In LabVIEW 8.0, when the <Enter> key or the <Return> key generates one of these events, LabVIEW returns **`<Enter>`** or **`<Return>`**, respectively, in the **VKey** data field.

(Mac OS) LabVIEW 8.0 accepts only <Control>-click for shortcut menus and does not receive the <Command>-click key combination. If you are emulating this behavior with an Event structure, modify your VIs to emulate the new behavior.

ListBox Properties

In LabVIEW 7.*x* and earlier, if you set the Top Row property of a listbox to a row that is below the bottom item of the listbox, LabVIEW pins the row to the last visible item. In LabVIEW 8.0, the number of visible items in the listbox does not limit the row number you can wire to this property.

LabVIEW 8.0 does not support the Double-Click property for single-column listboxes. Use the Get Double-Clicked Row method instead.

Owning VI Property

In LabVIEW 7.*x* and earlier, the Owning VI property returns a reference to the VI to which the object belongs. This reference keeps the VI in memory. In LabVIEW 8.0, the reference the Owning VI property returns does not keep the VI in memory. If the owning VI is removed from memory, this reference becomes invalid. Use the Open VI Reference function to obtain a reference to a VI that stays in memory until you explicitly close the reference.

Text Property

In LabVIEW 7.*x* and earlier, the Text property returns a string in normal display. In LabVIEW 8.0, the Text property returns a string in the same text display as the front panel object. For example, if you display a string control in password display, the Text property returns the string in password display.

TreeControl Properties

In LabVIEW 7.*x* and earlier, the Active Cell Properties:Cell Size:Height and Active Cell Properties:Cell Size:Width properties return 17 pixels for each line in the tree control. In LabVIEW 8.0, the Active Cell:Cell Size: Height and Active Cell:Cell Size:Width properties return 16 pixels for each line in the tree control.

VI Strings Methods

Strings that you export from previous versions of LabVIEW using the Export VI Strings method might not import properly in LabVIEW 8.0 when you use the VI Strings:Import method.

Deprecated Properties, Methods, and Events

LabVIEW 8.0 does not support the following properties, methods, and events.

Cursor Properties

LabVIEW 8.0 does not support the Cursor Lock Style property. Use the Cursor Mode property instead.

ListBox, Table, DigitalTable, and TreeControl Properties and Events

LabVIEW 8.0 does not support the Cell Foreground Color property for multicolumn listboxes. Use the Active Cell:Cell Font:Color property instead.

LabVIEW 8.0 does not support the Cell FG Color property for tables or digital tables. Use the Active Cell:Cell Font:Color property for tables and digital tables instead.

LabVIEW 8.0 does not support the Active Cell Properties:Foreground Color property for tree controls. Use the Active Cell:Cell Font:Color property instead.

LabVIEW 8.0 does not support the Drag, Drag?, Drop, and Drop? events in the TreeControl class. Use the Drag Ended, Drag Enter, Drag Leave, Drag Over, Drag Source Update, Drag Starting, Drag Starting?, and Drop events in the Control class instead.

NamedNumeric Properties

LabVIEW 8.0 does not support the Named Numeric Colors, Named Numeric Colors:BG Color, or Named Numeric Colors:Text Color properties for named numeric objects. Use the Text Colors, Text Colors:BG Color, and Text Colors:Text Color properties, respectively, instead.

Panel Properties

LabVIEW 8.0 does not support the Color property in the Panel class. If you use this property in LabVIEW 8.0, the property applies only to the upper-leftmost pane. Use the Pane Color property in the Pane class instead.

Subpanel Properties

In LabVIEW 8.0, use the pane of a subVI in a subpanel to configure the visibility of scroll bars for subpanel controls and to scale the front panel in subpanel controls.

LabVIEW 8.0 does not support the X Scrollbar Visible property for subpanel controls. Use the Horizontal Scrollbar Visibility property for panes instead.

LabVIEW 8.0 does not support the Y Scrollbar Visible property for subpanel controls. Use the Vertical Scrollbar Visibility property for panes instead.

LabVIEW 8.0 does not support the Scale Panel property for subpanel controls. Use the Set Scaling Mode method for panes instead.

VI Properties, Methods, and Events

LabVIEW 8.0 does not support the Front Panel Window:Auto Center property. Use the Front Panel:Center method instead.

LabVIEW 8.0 does not support the Front Panel Window:Size to Screen property. Use the Front Panel Window:State property instead.

LabVIEW 8.0 does not support the Front Panel Window:Origin property in the VI class. If you use this property in LabVIEW 8.0, the property applies only to the upper-leftmost pane. Use the Origin property in the Pane class instead.

LabVIEW 8.0 does not support the Front Panel Window:Show Scroll Bars property in the VI class. If you use this property in LabVIEW 8.0, the property applies only to the upper-leftmost pane. Use the Horizontal Scrollbar Visibility and Vertical Scrollbar Visibility properties in the Pane class instead.

LabVIEW 8.0 does not support the Get Front Panel Scaling Mode or Set Front Panel Scaling Mode methods in the VI class. If you use these methods in LabVIEW 8.0, the methods apply only to the upper-leftmost pane. Use the Get Scaling Mode and Set Scaling Mode methods in the Pane class instead.

LabVIEW 8.0 does not support the Mouse Down, Mouse Down?, Mouse Enter, Mouse Leave, Mouse Move, or Mouse Up events in the VI class. Use the Mouse Down, Mouse Down?, Mouse Enter, Mouse Leave, Mouse Move, and Mouse Up events in the Pane class, respectively, instead.

Application Item Menu Tags

The following application item menu tags were removed from LabVIEW:

- `APP_BUILD_STANDALONE_APP`
- `APP_DN_ASSEMBLY_REFS`
- `APP_EDIT_VI_LIBRARY`
- `APP_SAVE_WITH_OPTIONS`
- `APP_SHOW_CLIPBOARD`
- `APP_SRC_CODE_CTRL`

- `APP_SWITCH_EXEC_TARGET`
- `APP_UPDATE_VXI`
- `APP_VIEW_PRINTED_MANUALS`

When you use a run-time menu (`.rtm`) file that was saved in a previous version of LabVIEW and the file contains a deleted tag, LabVIEW 8.0 automatically removes the tag from the `.rtm` file when you save the file in the **Menu Editor** dialog box. The deleted application item tags are reserved by LabVIEW and you cannot use them as user tags.

HiQ Support

National Instruments does not support HiQ functionality in LabVIEW 8.0. If an application uses HiQ VIs, consider replacing them with the Mathematics and Signal Processing VIs. Refer to the *LabVIEW Help* for information about using the Mathematics and Signal Processing VIs.

Error List Window

In LabVIEW 7.*x* and earlier, the **VI List** section of the **Error list** window shows errors for all VIs in memory. In LabVIEW 8.0, the **Items with errors** section of the **Error list** window shows errors for all items in memory, such as VIs and libraries. If two or more items have the same name, this section shows the specific application instance for each ambiguous item. Refer to the *Working with Application Instances* section of this document for more information about application instances.

VI String File Syntax

LabVIEW 8.0 searches for a new set of tags, `<GROUPER></GROUPER>`, when you import VI string files by selecting **Tools»Advanced»Import Strings** or by using the VI Strings:Import method. This set of tags denotes front panel objects that are grouped together. Therefore, in LabVIEW 8.0, you cannot import VI string files saved in previous versions of LabVIEW.

LabVIEW 7.1 and earlier list listbox strings in the `<ITEMS>` section of its private data. LabVIEW 8.0 lists listbox strings in the `<STRINGS>` section of its private data. Also, in LabVIEW 7.1 and earlier, a listbox can have only one font, which LabVIEW lists in the `<LBLABEL>` section of its private data. In LabVIEW 8.0, the listbox can have multiple fonts, which LabVIEW lists in the `<CELL_FONTS>` section of its private data.

LabVIEW 7.1 and earlier list multicolumn listbox strings in its default data. However, the default data for a multicolumn listbox is an integer or array of integers. LabVIEW 8.0 lists multicolumn listbox strings in its private data.

LabVIEW 7.1 and earlier exports neither strings nor fonts for tree controls. LabVIEW 8.0 can export both tree control strings and fonts, and it exports them in the same format as the listbox and multicolumn listbox.

In LabVIEW 8.0, each line of an export file contains no more than two tags for private or default data. LabVIEW 8.0 also indents items once for each nesting level.

Complete the following steps to convert VI string files to the LabVIEW 8.0 format:

1. Import the VI string file in the previous version of LabVIEW.
2. Save the VI.
3. Load the VI in LabVIEW 8.0.
4. Select **Tools»Advanced»Export Strings** to save the VI string file in the LabVIEW 8.0 format.

Converting Type Descriptor Data to and from LabVIEW 7.*x*

The format in which LabVIEW stores type descriptors changed in LabVIEW 8.0. LabVIEW 7.*x* stores type descriptors in 16-bit flat representation. LabVIEW 8.0 stores type descriptors in 32-bit flat representation. This change eliminates the 64 KB size limitation of type descriptors.

LabVIEW 8.0 provides a mechanism for reading type descriptors written in LabVIEW 7.*x* and writing type descriptors that LabVIEW 7.*x* can read. The Flatten To String function has a **Convert 7.x Data** shortcut menu item. If you right-click the function and select this menu item, the function treats input data as if it were written for LabVIEW 7.*x*. When you select the **Convert 7.x Data** shortcut menu item and the **data string** output is wired, LabVIEW 8.0 places a red `7.x` glyph on the function to indicate that it is converting data to or from LabVIEW 7.*x* format. To avoid the conversion of data, select the **Convert 7.x Data** shortcut menu item again to remove the checkmark.

In LabVIEW 8.0, when you load a VI last saved in LabVIEW 7.*x* or earlier, LabVIEW 8.0 automatically sets the **Convert 7.x Data** attribute on the Flatten To String function. The function continues to operate as in LabVIEW 7.*x* and earlier. If you want a VI to use the LabVIEW 8.0 type descriptor format, right-click the Flatten To String function and select **Convert 7.x Data** from the shortcut menu to remove the checkmark. Use the LabVIEW 8.0 type descriptor format if VIs do not need to manipulate files that contain data written in LabVIEW 7.*x* or earlier and do not send or receive data to or from VIs running in LabVIEW 7.*x* or earlier. Support for the previous type descriptor format might be discontinued in future versions of LabVIEW.

Converting NaN Strings to Integer Types (Windows)

In LabVIEW 7.*x*, when you explicitly or implicitly convert `NaN` to an integer, the value becomes the smallest value for that integer data type. For example, converting `NaN` to a 16-bit signed integer produces the value –32,768, the smallest possible value for a 16-bit signed integer.

In LabVIEW 8.0, when you explicitly or implicitly convert `NaN` to an integer, the value becomes the largest value for that integer data type. For example, converting `NaN` to a 16-bit signed integer produces the value 32,767, the largest possible value for a 16-bit signed integer.

Constants Wired to Case Structures

In LabVIEW 7.*x* and earlier, you can keep subVIs in memory by wiring a constant to a Case structure and placing the subVI in a case that does not execute. For example, if you wire a TRUE constant to a Case structure and place a subVI in the FALSE case of the Case structure, LabVIEW loads the subVI along with the calling VI. LabVIEW 8.0 removes any code that does not execute. Therefore, if you load a VI in LabVIEW 8.0 that was saved in an earlier version of LabVIEW with a constant wired to a Case structure, LabVIEW changes the constant to a hidden control to maintain the behavior from the earlier version of LabVIEW.

Delaying Operating System Messages

In LabVIEW 7.*x*, LabVIEW processes operating system messages while running callback VIs for handling .NET and ActiveX events. In LabVIEW 8.0, LabVIEW delays the processing of operating system messages until the callback VI stops execution or until you load a modal dialog box. This delay allows callback VIs to execute without interruption and prevents LabVIEW from firing an event within another event, which can result in a deadlock state.

You cannot make synchronous calls to non-modal dialog boxes from a callback VI. You must asynchronously call a non-modal dialog box from a callback VI by invoking a Run VI method on the dialog and wiring a FALSE Boolean constant to the **Wait Until Done** input of the method.

In LabVIEW 7.*x*, LabVIEW processes operating system messages while running DLL or shared library functions. In LabVIEW 8.0, LabVIEW delays the processing of operating system messages until the end of calls to DLL functions or until you load a modal dialog box from the DLL. This delay allows DLL functions to execute without interruption and prevents

LabVIEW from calling the same DLL while a DLL function is running, which can result in a deadlock state.

If you use this default behavior, you cannot make synchronous calls to non-modal dialog boxes while a DLL runs. You must call a non-modal dialog box asynchronously from a DLL by invoking a Run VI method on the dialog and wiring a FALSE Boolean constant to the **Wait Until Done** input of the method.

You can choose whether to delay operating system messages in DLLs that you build. Navigate to the **Advanced** page of the **My DLL Settings** dialog box and remove the checkmark from the **Delay operating system messages in shared library** checkbox to process operating system messages while DLL functions run.

Resource Manager (Mac OS)

LabVIEW 7.*x* and earlier provide undocumented capabilities with which you can read and write Macintosh resource files. In LabVIEW 8.0, these methods do not exist. Utilities that make use of these undocumented capabilities do not work, and you therefore cannot read or write Macintosh resource files from VIs.

One- and Two-Button Dialog Boxes

In LabVIEW 7.*x* and earlier, you cannot abort programmatically a VI displaying a one-button dialog box or two-button dialog box. In LabVIEW 8.0, you can abort programmatically a VI displaying these dialog boxes by using the Abort VI method.

Property and Invoke Nodes

If you create an implicitly linked Property Node or Invoke Node from a cursor legend in LabVIEW 7.*x*, LabVIEW deletes the node when you open the VI in LabVIEW 8.0.

Updating Shared Libraries

If you build a shared library (DLL) in LabVIEW 7.*x* or earlier that links to `labview.lib`, link the shared library to `labviewv.lib` instead in LabVIEW 8.0. Refer to the *Using Shared Libraries in Multiple Versions of LabVIEW* section of this document for more information about using shared libraries in LabVIEW 8.0.

Margin Values for Printing

In LabVIEW 7.*x* and earlier, the **Margins** option on the **Printing** page of the **Options** dialog box uses centimeters for margin values. In LabVIEW 8.0, the **Margins** option uses millimeters for margin values.

Upgrading from LabVIEW 6.*x*

You might encounter the following compatibility issues when you upgrade to LabVIEW 8.0 from LabVIEW 6.*x*. Refer to the *Upgrading from LabVIEW 7.x* section of this document for information about other upgrade issues you might encounter.

Refer to the *LabVIEW Upgrade Notes* for each version of LabVIEW between versions 6.*x* and 8.0 at `ni.com/manuals` for more information about the new features and changes in each version.

Changes to the Waveform Data Type

In LabVIEW 7.0, the waveform data type uses the time stamp data type for the **t0** component rather than a double-precision, floating-point number. If you save data in the waveform data type to a file without including information about the data type in LabVIEW 6.*x*, you might encounter an error if you try to retrieve that data in LabVIEW 7.*x* and later.

In the LabVIEW 7.*x* and later, the Read Waveform from File VI converts the old waveform data type format in a file to the new waveform data type format. This VI displays a dialog box that prompts you to accept the conversion. In the LabVIEW Run-Time Engine, the Read Waveform from File VI cannot perform this conversion and returns an error instead. Refer to the National Instruments Web site at `ni.com/info` and enter the info code `exd9zq` for more information about migrating waveform data from LabVIEW 6.*x* to LabVIEW 7.*x* and later.

Serial Compatibility VIs

In LabVIEW 7.*x* and later, the Serial Compatibility VIs do not appear on the **Functions** palette. Use the VISA VIs and functions to build VIs that communicate with VXI devices.

In LabVIEW 7.*x* and later, LabVIEW does not use the `serpdrv` driver to communicate with the serial driver of the operating system. LabVIEW includes compatible VIs based on VISA. For new applications, use the VISA and Serial VIs and functions to control serial devices. Any VIs built in previous versions of LabVIEW that include Serial VIs continue to work in LabVIEW 7.1 and later.

If you reconfigured the mapping of port numbers to ports, you must specify a mapping to those ports. Use the set serial alias ports VI in the `labview\vi.lib\Instr\_sersup.llb` to specify the serial port mappings. Wire a string array to the **VISA Aliases** input of the VI and enter the port names you use in the input array. Each element in the array should correspond to a port. For example, if you configured port 0 to map to the VISA alias MySerialPort, enter `MySerialPort` as the first element of the **VISA Aliases** input array. You must call the set serial alias ports VI before you call the VISA Configure Serial Port VI.

Refer to the `labview\examples\instr\smplserl.llb` for examples of using the VISA VIs and functions to control serial instruments.

Default Data in Loops

In LabVIEW 6.0 and earlier, For Loops produce undefined data if the loop does not execute. In LabVIEW 6.1 and later, For Loops produce default data if you wire `0` to the count terminal of the For Loop or if you wire an empty array to the For Loop as an input with auto-indexing enabled. The loop does not execute, and any output tunnel with auto-indexing disabled contains the default value for the tunnel data type.

Remote Front Panel License

The LabVIEW Full Development System and the Application Builder include a remote front panel license that allows one client to view and control a front panel remotely. The LabVIEW Professional Development System includes a remote front panel license that allows five clients to view and control a front panel remotely.

You can upgrade the remote front panel license to support more clients.

Multiple Thread Allocation

LabVIEW 7.1 and later allocate more threads for executing VIs than in versions earlier than LabVIEW 7.1. Because of this change, you might encounter errors with multiple threads if you incorrectly mark Call Library Function Nodes as reentrant when the DLL you call is not actually reentrant. Refer to the *LabVIEW Help* for more information about the Call Library Function Node and reentrancy.

To change how LabVIEW allocates threads, use the threadconfig VI in the `labview\vi.lib\Utility\sysinfo.llb`. You also can disable reentrancy for VIs by selecting **File»VI Properties**, selecting **Execution** from the **Category** pull-down list, and removing the checkmark from the **Reentrant execution** checkbox.

Refer to the *LabVIEW Help* for more information about thread allocation.

Instrument Drivers

The LabVIEW package in LabVIEW 7.*x* and later does not include the LabVIEW Instrument Driver Library CD, which contains instrument drivers. Download instrument drivers from the National Instruments Instrument Driver Network at `ni.com/idnet`. The National Instruments Device Drivers CD includes NI-DAQ, NI-VISA, and other National Instruments drivers.

Units and Conversion Factors

In LabVIEW 7.*x* and later, you do not need to use the Convert Unit function to remove the extra unit after using the Compound Arithmetic function.

The unit conversion factors in LabVIEW 7.1 and later more closely match the guidelines published by the National Institute for Standards and Technology (NIST) in the *Guide for the Use of the International System of Units (SI)*. Also, the `calorie` unit now is `calorie (thermal)`, and `horse power` now is `horsepower (electric)`. The abbreviations for these units did not change. The following table details the changes in unit conversion factors between LabVIEW 6.1 and 8.0.

Table 1. Unit Conversion Factors

Unit	6.1 Definition	8.0 Definition
astronomical unit (AU)	149,498,845,000 m	149,597,900,000 m
British Thermal Unit (mean)	1055.79 J	1055.87 J
electron volt (eV)	1.602e–19 J	1.60217642e–19 J
foot-candle	10.764 lx	10.7639 lx
horse power versus horse power (electric)	745.7 W	746 W The new conversion is exact.
imperial gallon	4.54596 l	4.54609 l
light year	9.4605 Pm	9.46073 Pm
pound force	4.448 N	4.448222 N
rod	16.5 ft	5.029210 m
slug	32.174 lb	14.59390 kg
unified atomic mass (u)	1.66057e–27 kg	1.66053873e–27 kg

Defer Panel Updates Property

In LabVIEW 6.1 and earlier, LabVIEW waits until the Defer Panel Updates property is FALSE to redraw any front panel objects with pending changes. In LabVIEW 8.0, when you set this property to TRUE, LabVIEW redraws any front panel objects with pending changes and then defers all new requests for front panel updates. In some cases, this change can cause LabVIEW to redraw the changed elements of the front panel an extra time.

Data Ranges for Numeric Controls

In LabVIEW 6.1 and earlier, some numeric controls have a default minimum value of `0.00`, maximum value of `0.00`, increment value of `0.00`, and out of range action of **Ignore**. In LabVIEW 8.0, these numeric controls use the default data range values for the data type.

Coercion Dots and Type Definitions

In LabVIEW 6.1 and later, wires include information about type definitions, so you might notice more coercion dots on block diagrams. If you wire a type definition to a VI or function terminal that is not a type definition terminal, a coercion dot appears. A coercion dot also appears if you wire an output terminal that is a type definition to an indicator that is not a type definition. These coercion dots indicate where you are not using type definitions consistently in the VIs. In this case, coercion dots do not affect run-time performance.

Refer to the *LabVIEW Help* for information about using the Flatten To String function to flatten type definitions.

File Dialog Box Button Label

In LabVIEW 6.1 and earlier, the file dialog box that the File Dialog function displays has a button label of **Save** if the user can enter a new filename. Otherwise, the button label is **Open**. In LabVIEW 8.0, the button label on the file dialog box that the File Dialog Express VI displays is **OK** in all cases unless you change it. Use the **button label** input of the File Dialog Express VI to change the label of the button. If you use the File Dialog Express VI in an existing VI, consider reviewing the behavior of the VI to make sure the default label of **OK** is appropriate to the functionality of the VI.

Control Online Help Function

The **Path to the help file** input of the Control Online Help function now is required. You can wire a compiled help filename (`.chm` or `.hlp`) or the full path to a compiled help file to the input. If you wire only a compiled help filename, LabVIEW searches the `labview\help` directory for that file.

Run VI Method

In LabVIEW 7.1, if you set the **Auto Dispose Ref** input of the Run VI method to TRUE, LabVIEW automatically disposes the reference after the VI stops running. In LabVIEW 8.0, LabVIEW also immediately disposes the reference if the method returns an error. This behavior might break a VI at run time if part of the block diagram depends on the reference.

Displaying the Front Panel When Loaded

In LabVIEW 8.0, if you configure a VI to display the front panel when LabVIEW loads the VI and you load the VI using the VI Server, LabVIEW does not display the front panel. You must use the Front Panel:Open method to display the front panel programmatically.

Open VI Reference Function

In LabVIEW 6.1 and earlier, if you do not wire a value to the **options** parameter of the Open VI Reference function, LabVIEW creates a VI from a template if the template is not already in memory. If the template is in memory, LabVIEW opens a reference to the template. In LabVIEW 7.0 and later, if you use the Open VI Reference function to create a reference to a template that is already in memory, the function returns an error unless you specify `0x02` in the **options** parameter.

Exponential Representation

In LabVIEW 6.0 and earlier, the ^ operator represents exponentiation in the Formula Node. In LabVIEW 6.1 and later, the operator for exponentiation is `**`—for example, `x**y`. The ^ operator represents the bitwise exclusive or (XOR) operation.

IVI Configuration Store File

The IVI Configuration Store file format now requires that all names be case-sensitive. If you use logical names, driver session names, or virtual names in your application, make sure that the name you use matches the name defined in the IVI Configuration Store file exactly, without any variations in the case of the characters in the name.

Technical Support Form

In LabVIEW 7.*x* and later, the LabVIEW installation program does not install `techsup.llb`. Refer to the National Instruments Web site at `ni.com/support` to solve installation, configuration, and application problems and questions.

Upgrading from LabVIEW 5.*x*

You might encounter the following compatibility issues when you upgrade to LabVIEW 8.0 from LabVIEW 5.*x*. Refer to the *Upgrading from LabVIEW 6.x* and *Upgrading from LabVIEW 7.x* sections of this document for information about other upgrade issues you might encounter.

Refer to the *LabVIEW Upgrade Notes* for each version of LabVIEW between versions 5.*x* and 8.0 and to the *LabVIEW 5.1 Addendum* at `ni.com/manuals` for more information about the new features and changes in each version.

Converting Datalog Files

LabVIEW 8.0 checks the type definition of datalog files to determine if a conversion is necessary. If the datalog file is older than LabVIEW 6.0, or if the datalog file contains waveform data types and is older than LabVIEW 7.1, LabVIEW 8.0 converts the file for reading and appending. In all other cases, reading from the datalog file and appending to the datalog file does not convert the datalog file.

When you open a datalog file created in an earlier version of LabVIEW, LabVIEW 8.0 prompts you to convert the file to the LabVIEW 8.0 format. If you choose to convert it, LabVIEW replaces the datalog file with data converted to the new format. If you choose not to convert the file, LabVIEW 8.0 returns an error and does not open the file.

Note Make a backup copy of datalog files before converting if you plan to continue to use old data in LabVIEW 6.1 or earlier. You cannot revert to or read converted datalog files in LabVIEW 6.1 or earlier.

To automatically convert datalog files when you open them, add the following line to the LabVIEW preferences file:

```
silentDatalogConvert=True
```

(Mac OS) Add the following line:

```
silentDatalogConvert:True
```

(Linux) Add the following line:

```
labview.silentDatalogConvert:True
```

Set the preference to `False` if you do not want to convert datalog files automatically when you open them.

Connecting to the VI Server

You cannot make a connection to the VI Server of a LabVIEW 5.*x* application from a LabVIEW 8.0 client because the LabVIEW 5.*x* application does not recognize some aspects of the LabVIEW 8.0 VI Server protocol. However, you can connect to the VI Server of a LabVIEW 8.0 application from a LabVIEW 5.*x* client.

UDP Functions

In LabVIEW 6.*x* and later, the UDP VIs are not on the **Functions** palette but exist as compatibility VIs in the `labview\vi.lib\_oldvers\_oldvers.llb`. Use the built-in UDP functions for network communication instead.

Upgrading from LabVIEW 4.*x*

You might encounter the following compatibility issues when you upgrade to LabVIEW 8.0 from LabVIEW 4.*x*. Refer to the *Upgrading from LabVIEW 5.x*, *Upgrading from LabVIEW 6.x*, and *Upgrading from LabVIEW 7.x* sections of this document for information about other upgrade issues you might encounter.

Refer to the *LabVIEW Upgrade Notes* for each version of LabVIEW between versions 4.*x* and 8.0 and to the *LabVIEW 5.1 Addendum* at `ni.com/manuals` for more information about the new features and changes in each version.

Converting Boolean Data to and from LabVIEW 4.*x*

The format in which LabVIEW stores Boolean data changed between LabVIEW 4.*x* and LabVIEW 5.*x*. LabVIEW 4.*x* stores Boolean data in two bytes unless the data is in an array, in which case LabVIEW 4.*x* stores each Boolean element in a single bit. LabVIEW 8.0 stores a Boolean value in a single byte, regardless of whether it is in an array. This change enables more block diagram functions to support arrays of Boolean values and makes the behavior of these arrays more consistent with the behavior of arrays of numbers. The LabVIEW 8.0 Boolean data format affects data manipulation in Code Interface Nodes (CINs), but LabVIEW 8.0 provides compatibility for existing CINs.

If you write binary data that includes one or more Boolean values to a file in LabVIEW 4.*x*, its format is different than if you write the same data in LabVIEW 8.0. LabVIEW 8.0 provides a mechanism for reading binary data written in LabVIEW 4.*x* and writing binary data that LabVIEW 4.*x* can read. The Type Cast, Flatten To String, and Unflatten From String functions have a **Convert 4.x Data** shortcut menu item. If you right-click the function and select this menu item, the function treats binary data as if it were written for LabVIEW 4.*x*. To produce data formatted for LabVIEW 4.*x*, use the Write to Binary File, Flatten To String, or Type Cast function. To read data formatted for LabVIEW 4.*x*, use the Read from Binary File, Unflatten From String, or Type Cast function. When you select the **Convert 4.x Data** shortcut menu item, LabVIEW 8.0 places a red `4.x` glyph on the function to indicate that it is converting data to or from LabVIEW 4.*x* format. To avoid the conversion of data, select the **Convert 4.x Data** shortcut menu item again to remove the checkmark.

If you have several data files with Boolean values, you can create a VI that opens these files and writes the data to a new data file that LabVIEW 8.0 recognizes.

In LabVIEW 8.0, when you load a VI last saved in LabVIEW 4.*x* or earlier, LabVIEW 8.0 automatically sets the **Convert 4.x Data** attribute on the Write to Binary File, Read from Binary File, Type Cast, Flatten To String, and Unflatten From String functions. These functions continue to operate as before. If you want the VI to use the LabVIEW 8.0 Boolean data format, right-click the function and select **Convert 4.x Data** from the shortcut menu to remove the checkmark. Use the LabVIEW 8.0 Boolean data format if VIs do not need to manipulate files that contain Boolean data written in LabVIEW 4.*x* or earlier and do not send or receive data that contain Boolean data to or from VIs running in LabVIEW 4.*x* or earlier. Support for the previous Boolean data format might be discontinued in future versions of LabVIEW.

VI Control VIs

In LabVIEW 5.*x* and later, the VI Control VIs are not on the **Functions** palette but exist as compatibility VIs in the `labview\vi.lib\utility\victl.llb`. Use the VI Server functions Open VI Reference, Call By Reference Node, Property Node, and Invoke Node instead of the VI Control VIs.

Some of the error codes the VI Control VIs return are different in LabVIEW 8.0. In previous versions of LabVIEW, the VI Control VIs returned the error codes 7 and 1000. The VI Control VIs in LabVIEW 8.0 return the codes 1004 and 1003. If a VI built in LabVIEW 4.*x* checks for error codes 7 and 1000, you must modify the VI to work in LabVIEW 8.0.

DDE VIs (Windows)

In LabVIEW 5.*x* and later, the DDE VIs are not on the **Functions** palette but exist as compatibility VIs in the `labview\vi.lib\platform\dde.llb`.

Upgrading from LabVIEW 3.*x* or Earlier Versions

Refer to the National Instruments Web site at `ni.com` for information about using a VI conversion kit to upgrade from LabVIEW 3.*x* or earlier. Refer to the *Upgrading from LabVIEW 4.x*, *Upgrading from LabVIEW 5.x*, *Upgrading from LabVIEW 6.x*, and *Upgrading from LabVIEW 7.x* sections of this document for information about other upgrade issues you might encounter.

Refer to the *LabVIEW Upgrade Notes* for each version of LabVIEW between versions 3.*x* and 8.0 and to the *LabVIEW 5.1 Addendum* at `ni.com/manuals` for more information about the new features and changes in each version.

LabVIEW 8.0 Features and Changes

Refer to the *LabVIEW Help* for programming concepts, step-by-step instructions, and reference information about the LabVIEW 8.0 features. Access the *LabVIEW Help* by selecting **Help»Search the LabVIEW Help**.

Refer to the `readme.html` file in the `labview` directory for more information about the LabVIEW 8.0 features and changes.

Activating the LabVIEW License

(Windows) LabVIEW relies on licensing activation. You must activate a valid LabVIEW license before you can run LabVIEW. To activate LabVIEW, use the serial number you received as part of your installation package. You can activate the LabVIEW license in any of the following ways:

- During installation, enter the serial number and select to run the **Activation Wizard** at the end of installation.
- After you launch LabVIEW in evaluation mode, select **Activate** in the **Activation Startup** dialog box.
- While running LabVIEW in evaluation mode, select **Help»Activate LabVIEW**. The license activation does not take effect until you restart LabVIEW.

Refer to the *LabVIEW 8.0 Release Notes* for more information about licensing in LabVIEW.

Launching LabVIEW

When you launch LabVIEW, the **Getting Started** window appears, which you can use to create new VIs and projects, select among the most recently opened LabVIEW files, find examples, and access the *LabVIEW Help*. You also can access information and resources to help you learn about LabVIEW, such as specific manuals, help topics, and resources on the National Instruments Web site.

The **Getting Started** window disappears when you open an existing file or create a new file. You can display the window by selecting **View»Getting Started Window**. The **Getting Started** window also appears when you close all open front panels and block diagrams.

Logging In and Out of LabVIEW

You can login to the LabVIEW environment if there is a security domain configured in the network. The login information identifies LabVIEW when communicating with a remote VI Server that has an enforced access control list. You can programmatically login to and logout of LabVIEW and monitor user account changes from the block diagram using the NI Security:Login and NI Security:Logout methods and the NI Security User Change event, respectively.

Domain Account Manager

Select **Tools»Security»Domain Account Manager** to create and destroy domains, enforce domain policies, and manage user and group accounts within domains locally or remotely. The Domain Account Manager is especially useful when managing remote domains without being physically present at the remote machine that hosts the domain. The authentication protocol is secured with built-in data encryption and data integration.

Enhancements to the New Dialog Box

Click the **New** link in the **Getting Started** window to display the **New** dialog box. You can use the **New** dialog box to create new VIs, projects, project libraries, custom controls, shared variables, and so on. Place a checkmark in the **Add to project** checkbox to add the new item to an open project.

In LabVIEW 7.*x* and earlier, you can select the **Small dialog** or **Large dialog** buttons to reduce or enlarge the size of the **New** dialog box. LabVIEW 8.0 does not have the **Small dialog** or **Large dialog** buttons. However, if you resize the **New** dialog box, LabVIEW remembers the size of the dialog box the next time you access it.

In LabVIEW 7.*x* and earlier, the **New** dialog box shows a preview of the front panel and block diagram of a VI you select. In LabVIEW 8.0, a preview of the block diagram appears in the **Description** section of the **New** dialog box.

The **Create New** list does not contain the VI Template, Global Variable Template, and Control Template items from LabVIEW 7.*x* and earlier. On the front panel or block diagram of a VI or global variable, select **File»Save As** to display a file dialog box and select **Templates VIs** from the **Save as type** pull-down menu to save the item as a template. On the front panel of a custom control, select **File»Save As** to display a file dialog box, and select **Template Controls** from the **Save as type** pull-down menu to save the control as a template.

RT, FPGA, and PDA Targets

You must create a LabVIEW project to work with an RT, FPGA, or PDA target. Refer to the specific module documentation for more information about using projects with the LabVIEW Real-Time, FPGA, and PDA Modules.

Switching Execution Targets

In LabVIEW 7.*x* and earlier, you must select a target from the **Execution Target** pull-down menu on the **LabVIEW** dialog box to select a target in which you want to develop VIs. If you are currently working on a VI and you want to change the execution target, you can select **Operate»Switch Execution Target** and select the target to which you want to switch. In LabVIEW 8.0, you can work across execution targets using the project environment. In the **Project Explorer** window, right-click the project root and select **New»Targets and Devices** from the shortcut menu to display the **Add Targets and Devices** dialog box. If a target in the project supports other targets, you also can right-click the target and select **New»Targets and Devices** from the shortcut menu to add a target under the existing target. You can add and develop a VI under that target. You then can switch execution targets by switching between VIs under those targets. Refer to the module documentation for more information about switching execution targets for the LabVIEW Real-Time, FPGA, and PDA Modules.

LabVIEW Projects

Use projects to group together LabVIEW files and non-LabVIEW files, create build specifications, and deploy or download files to targets. A target is a device or machine on which a VI runs. When you save a project, LabVIEW creates a project file (`.lvproj`), which includes configuration information, build information, deployment information, references to files in the project, and so on.

You must use a project to build stand-alone applications and shared libraries. You also must use a project to work with an RT, FPGA, or PDA target. Refer to the specific module documentation for more information about using projects with the LabVIEW Real-Time, FPGA, and PDA Modules.

Project-style LabVIEW Plug and Play instrument drivers use the project and project library features in LabVIEW 8.0. You can use project-style drivers in the same way as previous LabVIEW Plug and Play drivers.

Project Explorer Window

Use the **Project Explorer** window to create and edit projects. Select **File»New Project** to display the **Project Explorer** window. You also can select **Project»New Project** or select **File»New** and then select **Empty Project** in the **New** dialog box to display the **Project Explorer** window.

The **Project Explorer** window includes the following items by default:

- **Project root**—Contains all other items in the **Project Explorer** window. The label of the project root includes the filename for the project.
  - **My Computer**—Represents the local computer as a target in the project.
  - **Dependencies**—Includes items that VIs under a target require.
  - **Build Specifications**—Includes build configurations for source distributions and other types of builds available in LabVIEW toolkits and modules. If you have the LabVIEW Professional Development System or Application Builder installed, you can use **Build Specifications** to configure stand-alone applications (EXEs), shared libraries (DLLs), and zip files. **(Windows)** You also can use **Build Specifications** to configure installers.

When you add another target to the project, LabVIEW creates an additional item in the **Project Explorer** window to represent the target. Each target also includes **Dependencies** and **Build Specifications**. You can add files under each target.

You can drag a VI from the **Project Explorer** window to the block diagram of another open VI to use the VI as a subVI.

Using a Project with Existing Files

You can add a group of existing files to a project. Select one of the following options for adding existing files to a project:

- If the existing files are part of a stand-alone application, select **Tools» Convert Build Script** to convert the application to a project. Refer to the *Changes from Previous Versions of the Application Builder* section of this document for more information about importing a build script.
- If the existing files are not part of a stand-alone application, create a new project and add the files to the **Project Explorer** window. Refer to the *Adding Items to a Project* section of this document for more information about adding files to a project.

Project Explorer Window Toolbars

Use the buttons on the Standard, Project, Build Specifications, and Source Control toolbars to perform operations in a project. The toolbars are available at the top of the **Project Explorer** window. You might need to resize the **Project Explorer** window to view all the toolbars.

You can show or hide toolbars by selecting **View»Toolbars** and selecting the toolbars you want to show or hide. You also can right-click an open area on the toolbar and select the toolbars you want to show or hide.

Adding Items to a Project

Use the **Project Explorer** window to add LabVIEW files, such as VIs and LLBs, as well as non-LabVIEW files, such as text files and spreadsheets, to a target in a project. You can create an organizational structure for items in a project.

You can add items under a target in a project in the following ways:

- Right-click a target or a folder under a target, select **Add File** from the shortcut menu, and select the file(s) you want to add from the file dialog box. You also can select the target, select **Project»Add To Project»Add File**, and select the file(s) you want to add from the file dialog box.
- Right-click a target and select **New»VI** from the shortcut menu to add a new, blank VI. You also can select **File»New VI** or **Project»Add To Project»New VI** to add a new, blank VI.
- Select the VI icon in the upper right corner of a front panel or block diagram window and drag the icon to the target.
- **(Windows and Mac OS)** Select an item or directory on disk and drag it to the target.

You also can add new LabVIEW files to a project from the **New** dialog box. Select **File»New** or **Project»Add To Project»New** to display the **New** dialog box. In the **New** dialog box, select the item you want to add and place a checkmark in the **Add to project** checkbox. If you have multiple projects open, select the project to which you want to add the item from the **Projects** list.

Adding Folders to a Project

Use the **Project Explorer** window to add a directory on disk as a folder in a project. Right-click a target or a folder under the target and select **Add Folder** from the shortcut menu to add the files from a directory on disk. Selecting a directory on disk creates project items that represent the contents of the entire directory, including files and contents of subdirectories. LabVIEW creates a new virtual folder in the project with the same name as the directory on disk.

You also can select a target and then select **Project»Add To Project»Add Folder** to add the files from a directory.

Note After you add a directory on disk to a project, LabVIEW does not automatically update the folder in the project if you make changes to the directory. If you make changes to the structure of subdirectories on disk, LabVIEW does not automatically update the subfolders in the project. If you want the folders in a project to match the directories on disk, you must manually update the folders.

You also can create new folders to organize items in a project. Right-click a target and select **New»Folder** from the shortcut menu to add a new folder under a target. You also can create a new subfolder by right-clicking an existing folder and selecting **New»Folder** from the shortcut menu.

Right-click the project root and select **New»Folder** from the shortcut menu to add a new folder to organize targets in a project.

Adding LLBs to a Project

You can add an LLB to a project as a folder or as a file. If you add an LLB as a folder, LabVIEW uses the filename of the LLB to name the folder and adds the VIs in the LLB as items within the new folder. Right-click a target or a folder under the target, select **Add Folder** from the shortcut menu, and use the file dialog box to navigate to the LLB you want to add. Adding or removing items from the folder in the **Project Explorer** window does not affect the `.llb` file on disk. Select **Tools»LLB Manager** to edit an LLB on disk.

You also can add an LLB to a project as a file. If you add an LLB as a file, the VIs in the LLB do not appear in the **Project Explorer** window.

Right-click a target or a folder under the target, select **Add File** from the shortcut menu, and use the file dialog box to navigate to the LLB you want to add. Select the open folder icon labeled **VI** with a single period to the right and click the **Select** button.

Removing Items from a Project

Use the **Project Explorer** window to remove items from a project. You can remove items in the following ways:

- Right-click the item you want to remove and select **Remove** from the shortcut menu.
- Select the item you want to remove and press the <Delete> key.
- Select the item you want to remove and click the **Delete** button on the **Standard** toolbar.

Note Removing an item from a project does not delete the corresponding item on disk.

Viewing Dependencies in a Project

Use **Dependencies** to view items that VIs under a target require. Each target includes **Dependencies**.

You cannot add items directly to **Dependencies**. LabVIEW adds dependencies for VIs under a target when you right-click **Dependencies** and select **Refresh** from the shortcut menu. For example, if you add a VI that includes a subVI to a target, LabVIEW adds the subVI to **Dependencies** when you select **Refresh**. However, if you add a dependent item under a target, the item does not appear under **Dependencies**. For example, if you add the VI and the subVI under the target, LabVIEW does not add the subVI to **Dependencies** when you select **Refresh**.

Dependencies include VIs, DLLs, and project libraries that a VI calls statically.

Note Items that a VI calls dynamically do not appear under **Dependencies**. You must add these items under a target to manage them in a project.

LabVIEW tracks subVIs recursively. LabVIEW does not track DLLs recursively. For example, if `a.vi` calls `b.dll` statically and `b.dll` calls `c.dll` statically, LabVIEW considers only `b.dll` a dependent item. To manage `c.dll` in the project, you must add `c.dll` under the target.

If a dependent item is part of a project library, LabVIEW adds the entire project library under **Dependencies**.

You cannot create new items under **Dependencies**. You cannot drag items from other places in the **Project Explorer** window to **Dependencies**.

To remove an item from **Dependencies**, right-click the item and select **Remove** from the shortcut menu. If you use this option to remove an item, the item reappears if you right-click **Dependencies** and select **Refresh** from the shortcut menu.

When you save a project, LabVIEW does not save the dependencies as part of the project. When you open a project, you must right-click **Dependencies** and select **Refresh** from the shortcut menu to view the dependencies.

Adding Targets to a Project (Windows)

Use the **Add Targets and Devices** dialog box to add a target or device to a project.

Note You must have a module or driver installed that supports targets to display this dialog box.

Right-click the project root and select **New»Targets and Devices** from the shortcut menu to display the **Add Targets and Devices** dialog box. If a target in the project supports other targets, you also can right-click the target and select **New»Targets and Devices** from the shortcut menu to add a target under the existing target. For example, if you have an NI PCI device installed on a computer, you can add the device under the **My Computer** target.

Deploying Files to a Target

In the **Project Explorer** window, open and run a VI under a target to deploy files to the target. When you run the VI, LabVIEW deploys the source files, build outputs, and hardware configuration to the target.

Using Palettes with Multiple Targets

LabVIEW loads a separate palette set for each target. If you have VIs open on multiple targets in a project, multiple palette sets exist in memory. When you switch between targets, LabVIEW maps each pinned palette to the corresponding palette in the active target. If a pinned palette does not have a corresponding palette, LabVIEW displays the top-level palette for the active target.

Working with Application Instances

LabVIEW creates an application instance, or an instance of LabVIEW, for each target in a project. When you open a VI from the **Project Explorer** window, the VI opens in the application instance for the target. LabVIEW also has a main application instance for open VIs that are not part of a project and VIs that you did not open from a project.

Use the application instance name that appears in the bottom left corner of the front panel and block diagram windows to identify which application instance a VI belongs to. If you have multiple projects open, the application instance name includes the project name followed by the target name, such as **Project 1.lvproj/My Computer**. Otherwise, the application instance name includes only the target name.

If you remove a VI from the **Project Explorer** window when the VI is open, the open VI remains in the same application instance. If you close and reopen the VI, LabVIEW opens the VI in the current application instance.

Editing VIs in Multiple Application Instances

You can open the same VI on disk in multiple application instances at the same time. For example, you can open the same VI from two different projects or from two different targets within a project.

If you edit a VI that is open in only one application instance and then open the VI in another application instance, the instance of the VI you just opened contains the latest changes from the edited instance of the VI. However, if you edit a VI that is already open in more than one application instance, LabVIEW does not automatically apply the changes to the VI in the other application instances. You cannot edit, run, or save the VI in the other application instances until the VI is the same in all application instances.

Use one of the following methods to make the VI the same in all application instances:

- Click the **Synchronize with Other Application Instances** button on the VI toolbar to apply changes to the VI in all application instances.
- Save the VI in the application instance that contains the changes. When you save the VI, LabVIEW automatically applies the changes to the VI in all other application instances.
- Undo edits to the VI in the application instance that contains the changes.

Note You cannot undo edits to the VI after you save or synchronize the VI.

You cannot edit a VI while it is running or reserved for execution in another application instance. If a VI is part of a project library, LabVIEW also temporarily locks the project library in all application instances when the VI runs. You cannot edit the project library while the VI is running. After the VI stops, LabVIEW unlocks the project library.

Using the Web Server for VIs in Projects

You can use the Web Server to view a VI or front panel that is part of a project. When you select **Operate»Connect to Remote Panel**, include the project name, project library, and target if applicable, in the **VI name**.

For example, if `MyVI.vi` resides in a project called `MyProject.lvproj` under target `My Computer`, enter the **VI name** as `MyProject.lvproj/My Computer/MyVI.vi`. If a project library called `MyLibrary` owns the VI, also include the project library in the **VI name**, as in `MyProject.lvproj/My Computer/MyLibrary:MyVI.vi`. If the VI is not in a project and a project library does not own the VI, enter the **VI name** without any additional information.

LabVIEW Project Libraries

LabVIEW project libraries are collections of VIs, type definitions, shared variables, palette menu files, and other files, including other project libraries. When you create and save a new project library, LabVIEW creates a project library file (`.lvlib`), which includes the properties of the project library and references to the files that the project library owns.

Project libraries are useful if you want to organize files into a single hierarchy of items, avoid potential VI name duplication, limit public access to certain files, limit editing permission for a collection of files, and set a default palette menu for a group of VIs.

You can view the structure of a project library from the **Project Explorer** window or in a stand-alone project library window. If the **Project Explorer** window is not open, navigate to a project library file to open it in the project library window.

You can create a project library in a LabVIEW project. Right-click **My Computer** and select **New»Library** from the shortcut menu. LabVIEW creates a project library file that appears under **My Computer**.

You also can create project libraries from folders in a project. From the **Project Explorer** window, right-click a folder and select **Convert to Library** from the shortcut menu.

Use project libraries to organize a virtual, logical hierarchy of items. A project library file does not contain the actual files it owns, unlike an LLB, which is a physical file that contains VIs. Files that a project library owns still appear individually on disk in the directories where you saved them. A project library might have a different organizational structure than its files on disk.

Use project libraries to qualify the names of VIs and other LabVIEW files. LabVIEW identifies VIs by filename. If you load a VI with the same name as a VI already in memory, LabVIEW uses the VI already in memory, an issue known as cross-linking. When a VI is part of a project library, LabVIEW qualifies the VI name with the project library name to avoid cross-linking. A qualified filename includes the filename and the owning project library filename.

For example, if you build a VI named `caller.vi` that includes a subVI named `init.vi` that `library1.lvlib` owns, you also can include a different subVI named `init.vi` that `library2.lvlib` owns and avoid cross-linking problems. The qualified filenames that LabVIEW records when you save `caller.vi` are `library1.lvlib:init.vi` and `library2.lvlib:init.vi` respectively.

Caution You must right-click a project library and select **Rename** from the shortcut menu to rename the project library. If you rename a project library outside LabVIEW, you might break the project library.

Right-click a project library and select **Properties** from the shortcut menu to limit public access to certain types of files, limit editing permission for project libraries, and set a palette menu file (`.mnu`) owned by the project library as the default palette menu for all VIs the project library owns.

After you select a default palette menu, you can right-click a subVI call to any VI that the project library owns and view the default palette for that project library from the shortcut menu.

Sharing Live Data Using Shared Variables

In LabVIEW 7.*x* and earlier, you use local and global variables to pass information in applications that you cannot connect with a wire. You select which type of variable to use depending on the scope of the application. LabVIEW 8.0 introduces a shared variable you can use to read and write live data among VIs in a project or across a network.

Shared variables combine the functionality of existing LabVIEW data transfer technologies, such as DataSocket, and you can manage the shared variables in the **Project Explorer** window. You can read and write shared variable data from the front panel or block diagram.

Shared variables are configured software items that can send data between VIs. Use shared variables to share data between VIs or between locations in an application that you cannot connect with wires. A shared variable can represent a value or an I/O point. You can change the scope, or **Variable Type**, or any of the shared variable properties without having to edit the block diagram of the VIs that use the shared variable.

Note You must configure firewalls and Network Address Translating (NAT) routers if you want to transmit shared variables through the firewalls or routers.

Creating Shared Variables

You must have a project open to create a shared variable. To add a shared variable to a project, right-click a target, a project library, or a folder within a project library in the **Project Explorer** window and select **New» Variable** from the shortcut menu.

Shared variables must be part of project libraries. If you create a shared variable from a target or folder that is not part of a project library, LabVIEW creates a new project library to own the shared variable.

Use the Shared Variable Refnum and the Variable properties to programmatically configure shared variables in a project. Create an indicator from the Variable Reference property to create a Shared Variable Refnum. You cannot make changes to single-process shared variables with this refnum.

Reading and Writing Shared Variable Values on the Front Panel

Use front panel data binding to read or write live data in a front panel object.

Note You can bind front panel objects only to network-published shared variables.

Drag a shared variable from the **Project Explorer** window to the front panel of a VI to create a control bound to the shared variable. You also can right-click a control, select **Properties** from the shortcut menu, and use the options on the **Data Binding** page to bind the control to a shared variable or to an NI Publish-Subscribe Protocol (NI-PSP) data item on the network.

Note You cannot right-click a control in LabVIEW 8.0 and select **Data Operations» DataSocket Connection** from the shortcut menu. Select **DataSocket** from the **Data Binding Selection** pull-down menu in the control **Properties** dialog box instead to connect front panel objects using the `dstp`, `opc`, `ftp`, `http`, and `file` protocols.

When you enable data binding for a control, changing the value of the control changes the value of the shared variable to which you bound the control.

Reading and Writing Shared Variable Values on the Block Diagram

A Shared Variable node is a block diagram object that points to the corresponding shared variable in the **Project Explorer** window. Use a Shared Variable node to read and write the value of the shared variable and to read the time stamp for the shared variable data. Drag a shared variable from the **Project Explorer** window onto the block diagram of a VI in the same project to create a Shared Variable node.

You also can select a Shared Variable node from the **Functions** palette and place it on the block diagram. To bind a Shared Variable node on the block diagram to a shared variable in the active project, double-click the Shared Variable node to display the **Select Variable** dialog box. You also can right-click the Shared Variable node and select **Select variable** from the shortcut menu.

Restricting Shared Variables to Single Writers

By default, multiple applications can write to a shared variable. However, you can set a network-published shared variable to accept changes in value from only one target at a time. Place a checkmark in the **Single Writer** checkbox on the **Variable** page of the **Shared Variable Properties** dialog box. This ensures that the shared variable write operation is not affected by another writer. The Shared Variable Engine restricts writing to a single VI on a single computer. The first writer that connects to the shared variable can write values, and any subsequent writers cannot. When the first writer disconnects, the next writer in the queue can write values to the shared variable. LabVIEW notifies writers that are not allowed to write to the shared variable.

Enabling Buffering for Shared Variables

If you use shared variables or a psp or dstp URL to share data programmatically, LabVIEW by default writes only the most recent value to all readers. When one client writes values to the server faster than another client reads them, newer values overwrite older, unprocessed values before the clients read them. If the reader does not receive a value before receiving the following value, the data is lost. This loss of unprocessed data can occur at the server or at the client. This loss of data might not be a problem if you are reading data and you want to receive only the most recent value written to the server. However, if you want to receive every shared variable value written to the server or if lossy transfers are unacceptable, you must buffer the data on the client. Place a checkmark in the **Use Buffering** checkbox on the **Variable** page of the **Shared Variable Properties** dialog box to enable buffering for the shared variable.

Managing Shared Variables

Select **Tools»Shared Variable»Variable Manager** to display the **Variable Manager** dialog box. Use this dialog box to edit, create, and monitor shared variables outside of the project environment. You also can manage the lifetime of the Shared Variable Engine with this dialog box.

Creating Source Distributions

VI distribution options, such as removing block diagrams and setting passwords, that were previously available in the **Save With Options** dialog box now are available when you build a source distribution.

Use **Build Specifications** in the **Project Explorer** window to build and create build specifications for source distributions. Right-click **Build Specifications** and select **New»Source Distribution** from the shortcut menu to display the **Source Distribution Properties** dialog box.

Changes from Previous Versions of the Application Builder

The Application Builder is integrated into the **Project Explorer** window. If you use the LabVIEW Base Package or Full Development System, you can purchase the Application Builder separately by visiting the National Instruments Web site at `ni.com` and entering the info code `rdlv21`.

Use **Build Specifications** in the **Project Explorer** window to build and create build specifications for stand-alone applications (EXEs), shared libraries (DLLs), and zip files. **(Windows)** You also can use **Build Specifications** to build and create build specifications for installers.

Build specifications are equivalent to `.bld` files in previous versions of the Application Builder, but they now are part of a LabVIEW project instead of separate files.

Note You must create a project to use the Application Builder tools. You can use the **Build Executable Wizard** from an open VI to create a project for an application. Select **Tools» Build Executable** to launch the **Build Executable Wizard**, which guides you through the process.

You can convert a `.bld` file into a build specification in a new project. Select **Tools»Convert Build Script** to navigate to and select the `.bld` file to convert. LabVIEW uses the file to create a project that contains the source files and build specifications.

LabVIEW 8.0 includes the following enhancements and changes to the Application Builder:

- Each type of build specification has a multipage dialog box that you can use to configure and edit build specification settings. Right-click the build specification and select **Properties** from the shortcut menu to access the dialog box.
- **(Windows)** Installer build specifications are separate from other types of build specifications. You can include more than one application or shared library build in an installer build. Right-click the installer build specification, select **Properties** from the shortcut menu, and navigate to the **Source Files** page to select the application, shared library, and source distribution builds to include in an installer.
- You can debug applications and shared libraries if you enable debugging in the build specification before you build the application or shared library. Right-click the build specification, select **Properties** from the shortcut menu, navigate to the **Advanced** page, and place a checkmark in the **Enable debugging** checkbox.

Note When you debug applications and shared libraries, you cannot debug reentrant front panels that an Open VI Reference function creates. You also cannot debug reentrant front panels that are entry points to LabVIEW-built shared libraries.

- **(Windows)** You can specify a version number in application and shared library build specifications. Right-click an application build specification, select **Properties** from the shortcut menu, and navigate to the **Application Information** page to set a version number for the application. Right-click a shared library build specification, select **Properties** from the shortcut menu, and navigate to the **Shared Library Information** page to set a version number for the shared library.
- **(Windows)** You can include installers or drivers for National Instruments products in installer builds. Right-click the installer build specification, select **Properties** from the shortcut menu, and navigate to the **Additional Installers** page to add drivers, LabVIEW Run-Time Engine support, and other National Instruments product installers to the installer build.
- You can configure run-time languages for an application or shared library in the build specification. Right-click the application or shared library build specification, select **Properties** from the shortcut menu, and navigate to the **Run-Time Languages** page to set language preferences for the application or shared library.

- The LabVIEW Run-Time Engine is multilingual, so you do not have to include multiple versions of it with an application, shared library, or installer.
- **(Windows)** You can configure an installer build specification to include registry entries and to create multiple shortcuts for files in the installer. Right-click the installer build specification, select **Properties** from the shortcut menu, and navigate to the **Registry** page to create and configure custom registry keys. Use the **Shortcuts** page to create and configure shortcuts.
- If you build an installer for an application or shared library that uses the Storage VIs, you must include the NI USI installer. Right-click the installer build specification, select **Properties** from the shortcut menu, navigate to the **Additional Installers** page, and select **NI USI** from the **National Instruments Installers to Include** list to include the NI USI installer.

New Palette Organization

LabVIEW 8.0 includes the following enhancements to the **Controls** and **Functions** palettes:

- Palette items are organized according to categories. The **Category (Standard)** and **Category (Icons and Text)** formats replace the **Standard** and **Standard (Icons or Text)** formats, respectively. The following tables list top-level palettes on the **Controls** palette from LabVIEW 7.1 and their category location in LabVIEW 8.0.

Controls Palette

Table 2. Controls Palette Organization

7.1 Palette	8.0 Category
Express	Express
Numeric	Modern
Boolean	Modern
String & Path	Modern
Array & Cluster	Modern
List & Table	Modern
Graph	Modern
Ring & Enum	Modern
Containers	Modern

Table 2. Controls Palette Organization (Continued)

7.1 Palette	8.0 Category
I/O	Modern
Refnum	Modern
Decorations	Modern
Dialog Controls	System
Classic Controls	Classic

Functions Palette

Table 3. Functions Palette Organization

7.1 Palette	8.0 Category
Express	Express
Structures	Programming
Numeric	Programming
Boolean	Programming
String	Programming
Array	Programming
Cluster	Programming
Comparison	Programming
Time & Dialog	Programming
File I/O	Programming
NI Measurements	Measurement I/O
Waveform	Programming
Analyze	Mathematics and Signal Processing
Instrument I/O	Instrument I/O
Application Control	Programming
Graphics & Sound	Programming
Communication	Connectivity and Data Communication
Report Generation	Programming

Table 3. Functions Palette Organization (Continued)

7.1 Palette	8.0 Category
Advanced	Connectivity and Data Communication
Decorations	Programming

- In the **Category (Standard)** and **Category (Icons and Text)** formats, you can right-click a category and select **Move this Category Up** or **Move this Category Down** from the shortcut menu to change the order in which categories appear on the palettes. You also can click the double lines to the left of a category and drag the arrow to where you want the category to appear.
- By default, LabVIEW installs with an abridged palette view, unless you are upgrading and were using the **Advanced** palette view in a previous version of LabVIEW. The abridged palette view hides some advanced categories at the top level of the palettes. Click the **View** button on the palette toolbar and select **Always Visible Categories» Show All Categories** from the shortcut menu to display all categories on the **Controls** or **Functions** palette.
- You can change the format of the current palette or of all palettes. Click the **View** button on the **Controls** or **Functions** palette toolbar and select a format from the **View This Palette As** shortcut menu to change the format for the current palette. Select **Tools»Options** and select the **Controls/Functions Palettes** page to select a format for all palettes.
- The **Tree format**—displays palette items as text in a tree control. Click the **View** button on the **Controls** or **Functions** palette toolbar and select **View This Palette As»Tree** from the shortcut menu to set the current palette to the tree format.
- In the **Text** and **Tree** formats, click the **View** button on the **Controls** or **Functions** palette toolbar and select **Sort Alphabetically** from the shortcut menu to sort items on the same level alphabetically.
- Use the **Favorites** category to group together items on the **Functions** palette that you access frequently. You can add items to the **Favorites** category using the **Category (Standard)**, **Category (Icons and Text)**, **Icons**, and **Icons and Text** formats. On a pinned **Functions** palette, right-click an object and select **Add Item to Favorites** from the shortcut menu to add the object to the **Favorites** category. In the **Category (Standard)** and **Category (Icons and Text)** formats, you also can expand a palette to display a subpalette, right-click the title of the subpalette, and select **Add Subpalette to Favorites** from the shortcut menu.

Moved and Renamed Palettes

The following palettes moved or were renamed in LabVIEW 8.0.

- On the **VISA Advanced** palette, the **Interface Specific** and **High Level Register Access** palettes have been renamed to **Bus/Interface Specific** and **Register Access**, respectively.
- The VIs from the **Advanced Formula Parsing** palette now are on the **Formula Parsing** and **1D & 2D Evaluation** palettes.
- The VIs from the **Cluster** and **Variant** palettes now are on the **Cluster & Variant** palette.
- The **Array Operations**, **Curve Fitting**, **Formulas**, **Logarithmic**, and **Special and Numeric Functions** palettes have been renamed to the **Signal Operation**, **Fitting**, **Scripts & Formulas**, **Exponential Functions**, and **Elementary & Special Functions** palettes, respectively. Some of the Curve Fitting VIs now are on the **Interp & Extrap** palette.
- The **Express Numeric Constants** palette has been renamed to the **Express Math & Scientific Constants** palette.
- The **Additional Numeric Constants** palette has been renamed to the **Math & Scientific Constants** palette.
- The **Express Logarithmic** palette has been renamed to the **Express Exponential Functions** palette.
- The VIs from the **Array Operations PtByPt** palette now are on the **Geometry PtByPt**, **Polynomial PtByPt**, **Probability & Statistics PtByPt**, and **Signal Operation PtByPt** palettes.
- The VIs from the **Curve Fitting PtByPt** palette now are on the **Fitting PtByPt** and **Interpolation PtByPt** palettes.
- The **Advanced** palette has been renamed to the **Libraries & Executables** palette.
- The VIs from the **Time & Dialog** palette now are on the **Timing** and **Dialog & User Interface** palettes.
- The VIs from the **Frequency Domain** palette now are on the **Spectral Analysis** and **Transforms** palettes.
- The VIs from the **Frequency Domain PtByPt** palette now are on the **Spectral Analysis PtByPt** and **Transforms PtByPt** palettes.
- The VIs from the **Time Domain** palette now are on the **Signal Operation** and **Integ & Diff** palettes.
- The VIs from the **Time Domain PtByPt** palette now are on the **Signal Operation PtByPt** and **Integral & Differential PtByPt** palettes.

- The **Timed Loop** palette has been renamed to the **Timed Structures** palette.
- The **All VIs and Functions** palette is not on the **Functions** palette.

Palette Editing Enhancements

LabVIEW 8.0 includes the following enhancements to editing palettes:

- In LabVIEW 7.*x* and earlier, you can create or edit a custom view of the **Controls** or **Functions** palettes. LabVIEW 8.0 does not support custom palette views. You can edit a palette set without using a custom palette view. If you are working on multiple targets, LabVIEW loads a palette set for each target. To edit the palette set for a specific target, display the **Edit Controls and Functions Palette Set** dialog box from a VI on the target whose palette set you want to modify. Select **Tools»Advanced»Edit Palette Set** to display the **Edit Controls and Functions Palette Set** dialog box.
- LabVIEW saves changes that you make to a palette set to the `8.0\Palettes` folder in the default data directory.
- To hide an item that is synchronized with a directory, right-click the item and select **Hide Synchronized Item** from the shortcut menu.
- If an item is synchronized to a directory, you can display the path to the source directory by right-clicking the item and selecting **Display Synchronization Path** from the shortcut menu.
- If a row or column appears at the end of a palette and contains only synchronized items that are hidden and/or items that LabVIEW cannot find, the row or column is visible only when editing palettes.
- Empty subpalettes or subpalettes that contain only items that LabVIEW is unable to find are visible only when editing palettes.

Menu Reorganization

LabVIEW has new menus and many items have moved or been renamed. The following items have changed in the LabVIEW menus:

- **File**—Added menu items for basic project operations, such as creating, opening, saving, and closing projects.
- **Edit**—Moved several **Operate** menu items to the **Edit** menu and added a new **Select All** item.
- **View**—Added a new menu that includes items for displaying windows and palettes, browsing classes, and displaying the **Project Explorer** window. Several **Window** menu items now are part of the **View** menu. Because the **Browse** menu has been removed, the **View** menu includes items from the **Browse** menu. Select **View»Browse Relationships** to find the **Browse** menu items.

- **Project**—Added a new menu that includes items for adding VIs and files to projects, building build specifications, and viewing project information.
- **Operate**—Added or moved items for debugging VIs to the **Operate** menu. Because the **Browse** menu has been removed, the **Operate** menu includes items from the **Browse** menu.
- **Tools**—Added several items to the **Tools** menu. Many of the existing **Tools** menu items have been rearranged into different submenus.
- **Window**—Removed items for viewing palettes, the **Navigation** window, and the **Error list** window. These items are part of the **View** menu.
- **Help**—Added item for finding instrument drivers and activating LabVIEW.

Note Some menu items are available only on specific operating systems, with specific LabVIEW development systems, when an item in the **Project Explorer** window is selected, or when a VI is selected.

Refer to the *LabVIEW Help* for more information about new menu locations, names, and instructions.

Using Source Control

If you have a third-party source control provider installed, such as Perforce or Microsoft Visual SourceSafe, you can use source control within a LabVIEW project or with individual VIs that are not part of a project. You can perform source control operations on VIs, files within a project, and project files, such as project libraries. LabVIEW provides an infrastructure to connect to external third-party source control providers. You must install a third-party source control provider to use source control in LabVIEW. **(Windows)** LabVIEW integrates with any source control provider that supports the Microsoft Source Code Control Interface. **(Mac OS and Linux)** You can use the Perforce command-line interface.

LabVIEW 8.0 includes the following changes to source control functionality.

Note Source control is available only in the LabVIEW Professional Development System.

- The built-in source control provider from LabVIEW 7.*x* and earlier is not available in LabVIEW 8.0. Refer to the *Migrating from the LabVIEW Built-in Source Control Provider* section for more information about migrating source control files from earlier versions of LabVIEW.

- After you select and install a third-party source control provider, you must configure LabVIEW to work with that provider. Select **Tools» Source Control»Configure Source Control** to display the **Source Control** page of the **Options** dialog box.
- You can use source control within LabVIEW to check out files for editing, check in edited files, get the latest version of a file from source control, and compare a file to the latest version in source control. Select **Tools»Source Control** and the source control operation you want to perform.
- You can view the source control history of a file. Select **Tools»Source Control»Show History** to display a provider-specific dialog box that contains the file revision history in source control.
- Use the Source Control VIs to perform source control operations programmatically.
- You cannot create Perforce configuration files in LabVIEW.

Migrating from the LabVIEW Built-in Source Control Provider

The built-in source control provider from LabVIEW 7.*x* and earlier is not available in LabVIEW 8.0. If you want to use source control in LabVIEW, you must select a third-party source control provider. If you used the built-in provider in previous versions, you must migrate the files to another provider to use source control in LabVIEW. Refer to the National Instruments Web site at `ni.com` and enter the info code `rdbp01` for the most current list of third-party source control providers supported in LabVIEW.

When you migrate files to a new source control provider, you lose the revision history stored in the built-in provider. You cannot transfer the previous versions of the files to the new provider.

Complete the following steps to migrate files from the built-in source control provider to a third-party source control provider.

1. In the previous version of LabVIEW, make sure that the files included in the LabVIEW built-in source control provider are checked in by all users.
2. On the computer where you want to add the files to the new source control provider, use the built-in provider to get the latest versions of all the files.
3. Use the built-in provider to check out the files from source control.
4. In the third-party source control provider, configure the settings you want for the new source control project.

5. Configure LabVIEW to work with the third-party source control provider.
6. Create a LabVIEW project. Add the files included in the built-in provider to the project. When LabVIEW prompts you, add the files to source control. You also can add the files directly in the third-party provider.

VI and Function Enhancements

LabVIEW 8.0 introduces the following enhancements and changes to VIs and functions.

Analyze VIs Enhancements

The VIs from the **Analyze** palette in LabVIEW 7.*x* and earlier now are in three separate palettes, **Mathematics**, **Signal Processing**, and **Programming**. The icons for many of these VIs changed. A yellow banner on the top of each icon indicates the subpalette in which the VI is located.

Improved and Changed VIs

The following Mathematics, Signal Processing, and Programming VIs are improved or changed in LabVIEW 8.0.

Mathematics Palette

The **Fitting** VIs have the following improvements and changes:

- The **method** input of the Exponential Fit, Exponential Fit Coefficients, Linear Fit, and Linear Fit Coefficients VIs specifies the fitting method to use. You can use the Least Square, Least Absolute Residual, or Bisquare fitting method. These VIs also have a **tolerance** input that determines when to stop the iterative adjustment of output parameters if the method you choose requires iterations. The **Weight** input of these VIs specifies the weights of the observed data values.
- The **residue** output of the Exponential Fit and Linear Fit VIs returns the weighted mean error of the fitted model.
- The **refine?** input of the Exponential Fit and Exponential Fit Coefficients VIs specifies whether to further refine the **amplitude** and **damping** of the fitted model.
- The Exponential Fit VI does not have the **Standard Deviation** input or the **mse** output from LabVIEW 7.*x* and earlier.
- The Linear Fit VI does not have the **mse** output from LabVIEW 7.*x* and earlier.
- The **Weight** input of the General LS Linear Fit VI specifies the weights of the observed data values. This VI also has a new algorithm, **SVD for Rank Deficient H**, to compute the best fit. Use this algorithm

only if the input matrix is rank deficient and all other algorithms are unsuccessful. This VI does not have the **Standard Deviation** input from LabVIEW 7.*x* and earlier.

- The **Coefficient Constraint** input of the General Polynomial Fit VI specifies the constraints on polynomial coefficients of certain orders. The **Polynomial Fit Coefficients** output of this VI now is called **Polynomial Coefficients**.

The **Integ & Diff** VIs have the following improvements and changes:

- The Numeric Integration VI can perform 2D and 3D numeric integration.

The **Linear Algebra** VIs have the following improvements and changes:

- In LabVIEW 7.*x*, the **eigenvector** output on the Eigenvalues and Vectors VI was normalized so that its largest component is always unified. In LabVIEW 8.0, the **eigenvector** output is normalized so that its Euclidean norm equals 1.
- The QR Decomposition VI replaces the QR Factorization VI. The QR Decomposition VI has two new inputs. **pivot?** specifies whether this VI decomposes the input matrix **A** with column pivoting. **Q option** specifies how this VI generates the orthogonal matrix **Q**. This VI can generate a full-size, economy-size, or no orthogonal matrix. The **P** output of this VI returns the permutation matrix.
- The QZ Decomposition VI has two new inputs. **decomposition type** specifies whether to perform a Generalized Hessenberg or a Generalized Schur decomposition. **order** specifies how to order the generalized eigenvalues, **Alpha** and **Beta**.
- The **order** input of the Schur Decomposition VI specifies how to order the **Eigenvalues** and the corresponding **Schur Form** and **Schur Vectors**.

The **Polynomial** VIs have the following improvements and changes:

- The Polynomial Roots VI can find the roots of a complex polynomial. This VI also provides the **Advanced Refinement** root-finding option.
- The **number of zero coefficients** output of the Remove Zero Coefficients VI returns the number of trailing near-zero coefficients that LabVIEW removed.

The **Probability and Statistics** VIs have the following improvements and changes:

- The Mode VI is polymorphic to perform both unimodal and multimodal analysis. The **intervals** input of this VI can accept non-positive values. If **intervals** is non-positive, the Mode VI calculates only the exact mode(s) of the input sequence.

- The **X** input of the Standard Deviation and Variance VI now is recommended instead of required.

The **Rational Polynomial** VIs have the following improvements and changes:

- The **c[0..n+m]** input of the Pade Approximation VI now is called **Derivatives**. Also, the **A[0..m]** output now is called **Numerator**, and the **B[0..n]** output now is called **Denominator**.

Signal Processing Palette

The **Advanced FIR Filtering** VIs have the following improvements and changes:

- The **window parameter** input of the FIR Windowed Coefficients VI specifies particular parameter values for Kaiser, Gaussian, and Dolph-Chebyshev windows. The data type of the **window** input for this VI changed from enumerated type to 32-bit unsigned integer numeric.

The **Advanced IIR Filtering** VIs have the following improvements and changes:

- The Cascade –> Direct Coefficients VI now is called the Cascade To Direct Coefficients VI.
- The IIR Cascade Filter, IIR Cascade Filter with I.C., IIR Filter, and IIR Filter with I.C. VIs are polymorphic and support complex inputs.

The **Filters** VIs have the following improvements and changes:

- The **window parameter** input of the FIR Windowed Filter VI specifies particular parameter values for Kaiser, Gaussian, and Dolph-Chebyshev windows. The data type of the **window** input for this VI changed from enumerated type to 32-bit unsigned integer numeric.
- All VIs on the **Filters** palette, except for the Median Filter VI, are polymorphic and support complex inputs.

The **Signal Generation** VIs have the following improvements and changes:

- The Binary MLS VI has an **error code** output.
- The **exclude end?** input of the Ramp Pattern VI specifies whether to include the ending value in the generated array.
- The **signal type** input of the Signal Generator by Duration VI now is recommended instead of required.

The **Signal Generation PtByPt** VIs have the following improvements and changes:

- The Sawtooth Wave PtByPt VI was optimized to include an updated formula, as follows:

  sawtooth wave = amplitude × sawtooth(*q*)

  where

$$\text{sawtooth}(q) = \begin{cases} \dfrac{q}{180} & \text{if } q < 180 \\ \dfrac{q}{180} - 2 & \text{else} \end{cases}$$

  and

  q =(360 · **f** · **time**) + **phase**) mod (360), phase in degrees.

The **Signal Operation** VIs have the following improvements and changes:

- The Autocorrelation, Convolution, and CrossCorrelation VIs are polymorphic and support 2D-array real and complex inputs.
- The **normalization** input of the 1D instances of the Autocorrelation and CrossCorrelation VIs specifies the normalization method to use to compute the autocorrelation and cross correlation, respectively.
- The **output size** input of the 2D instances of the Convolution VI specifies the size of the convolution.
- The Normalize Vector VI and the Normalize Matrix VI comprise the Normalize polymorphic VI.
- The Quick Scale 1D VI and the Quick Scale 2D VI comprise the Quick Scale polymorphic VI. The **Yij=Xij/Max{X}** parameter of the Quick Scale 2D instance of the Quick Scale VI now is called **Yij=Xij/Max|X|**.
- The Resample (constant to constant) and Resample (constant to variable) VIs support complex inputs.
- The Scale 1D VI and the Scale 2D VI comprise the Scale polymorphic VI.
- The Unit Vector VI is polymorphic and has two new inputs. **norm type** indicates what type of norm is used to compute the norm. **user defined norm** specifies the norm type if **norm type** is **User Defined**.
- The Zero Padder VI is polymorphic and supports complex inputs. The **only non-power of 2?** input of this VI specifies whether to double the size of the input array if the array size already is a valid power of 2.

The **Spectral Analysis** VIs have the following improvements and changes:

- The Power Spectrum VI is polymorphic and supports complex inputs.
- The STFT Spectrogram VI now is called the STFT Spectrograms VI and has two new inputs. **time-freq sampling info** specifies the density to use to sample the signal in the joint time-frequency domain and defines the size of the resulting 2D time-frequency array. **window info** specifies information about the window you want to use to compute the Short-Time Fourier Transform. You now can use the Blackman-Harris, Exact Blackman, Flat Top, 4 Term B-Harris, 7 Term B-Harris, Low Sidelobe, Blackman Nuttall, and Triangle windows. This VI does not have the **time increment**, **window length**, or **window selector** inputs from LabVIEW 7.*x* and earlier.

The **Transforms** VIs have the following improvements and changes:

- The FFT and Inverse FFT VIs support 2D-array real and complex inputs.
- The **shift?** input of the FFT and Inverse FFT VIs specifies whether to shift the DC component to the center of the Fast Fourier Transform.
- The **FFT size** input of the 1D instances of the FFT VI specifies the length of the FFT you want to perform.

The **Transforms PtByPt** VIs have the following improvements and changes:

- The Real FFT PtByPt VI and the Complex FFT PtByPt VI comprise the FFT PtByPt polymorphic VI.
- The Inverse Real FFT PtByPt VI and the Inverse Complex FFT PtByPt VI comprise the Inverse FFT PtByPt polymorphic VI.

The **Waveform Conditioning** VIs have the following improvements and changes:

- The Scaled Window VI supports complex inputs. The **window parameter** input of this VI specifies particular parameter values for Kaiser, Gaussian, and Dolph-Chebyshev windows. The data type of the **window** input for this VI changed from enumerated type to 32-bit unsigned integer numeric.

The **Waveform Measurements** VIs have the following improvements and changes:

- The data type of the **window** input changed from enumerated type to 32-bit unsigned integer numeric for the Cross Spectrum (Mag-Phase), Cross Spectrum (Real-Im), FFT Power Spectral Density, FFT Power Spectrum, FFT Spectrum (Mag-Phase), FFT Spectrum (Real-Im), Frequency Response Function (Mag-Phase), and Frequency Response Function (Real-Im) VIs.

The **Windows** VIs have the following improvements and changes:

- The **ratio** input of the Cosine Tapered Window VI specifies the ratio of the length of the tapered section to the length of the entire signal. The **Cosine Tapered{X}** output of this VI now is called **Windowed X**.
- The **window parameter** of the Scaled Time Domain Window VI specifies parameters for the window you want to apply. This VI also has a new **error** output. The **Waveform** input, **Windowed Waveform** output, and **window constants** output of this VI now are called **X**, **Windowed X**, and **window properties**, respectively. The Scaled Time Domain Window VI now also can apply a Blackman-Nuttall, Triangle, Kaiser, Dolph-Chebyshev, or Gaussian window. The data type of the **window** input of this VI changed from enumerated type to 32-bit unsigned integer numeric.
- All VIs on the **Windows** palette except for the Window Properties VI are polymorphic and support complex inputs.

Programming Palette

The **Comparison** functions have the following improvements and changes:

- The Equal? and Not Equal? comparison functions can determine whether two VI Server references refer to the same object.

New VIs and Functions

LabVIEW 8.0 introduces the following new Mathematics, Signal Processing, and Programming VIs.

Mathematics Palette

The **Advanced Curve Fitting** subpalette contains the following new VIs:

- Exponential Fit Intervals VI
- Gaussian Peak Fit Coefficients VI
- Gaussian Peak Fit Intervals VI
- Goodness of Fit VI
- Linear Fit Intervals VI
- Logarithm Fit Coefficients VI
- Logarithm Fit Intervals VI
- Power Fit Coefficients VI
- Power Fit Intervals VI
- Remove Outliers VI

The **Differential Equations** subpalette contains the new ODE Solver VI.

The **Discrete Math** subpalette contains the following new VIs:

- Gcd VI
- Lcm VI
- Permute VI

The **Exponential Functions** subpalette contains the new y-th root of x function.

The **Fitting** subpalette contains the following new VIs:

- Cubic Spline Fit VI
- Gaussian Peak Fit VI
- Logarithm Fit VI
- Nonlinear Curve Fit VI
- Power Fit VI

The **Geometry** subpalette contains the following new VIs:

- 2D Cartesian Coordinate Shift VI
- 2D Cartesian Coordinate Rotation VI
- 3D Cartesian Coordinate Shift VI
- 3D Cartesian Coordinate Rotation (Euler) VI
- 3D Cartesian Coordinate Rotation (Direction) VI
- 3D Coordinate Conversion VI
- Direction Cosines to Euler Angles VI
- Euler Angles to Direction Cosines VI

The **Hyperbolic Functions** subpalette contains the following new functions:

- Hyperbolic Cosecant function
- Hyperbolic Cotangent function
- Hyperbolic Secant function
- Inverse Hyperbolic Cosecant function
- Inverse Hyperbolic Cotangent function
- Inverse Hyperbolic Secant function

The **Hypothesis Testing** subpalette contains the following new VIs:

- Correlation Test VI
- Sign Test VI
- T Test VI

- Wilcoxon Signed Rank Test VI
- Z Test VI

The **Integ & Diff** subpalette contains the new Lobatto Quadrature VI.

The **Interp & Extrap** subpalette contains the following new VIs:

- Create Interpolating Polynomial VI
- Create Mesh Grid (2D) VI
- Evaluate Interpolating Polynomial VI
- Hermite Interpolation 1D VI
- Interpolate 1D VI
- Interpolate 1D Fourier VI
- Interpolate 2D VI
- Search Ordered Table VI
- Spline Interpolation 1D VI

The **Numeric** subpalette contains the following new functions:

- Machine Epsilon constant
- Square function

The **Optimization** subpalette contains the following new VIs:

- Constrained Nonlinear Optimization VI
- Quadratic Programming VI
- Unconstrained Optimization VI

The **Polynomial** subpalette contains the new Polynomial Plot VI.

The **Probability** subpalette contains the following new VIs:

- Continuous CDF VI
- Continuous Inverse CDF VI
- Continuous Moments VI
- Continuous PDF VI
- Continuous Random VI
- Discrete CDF VI
- Discrete Inverse CDF VI
- Discrete Moments VI
- Discrete PF VI
- Discrete Random VI

The **Probability and Statistics** subpalette contains the following new VIs:

- Correlation Coefficient VI
- Correlation Coefficient (Kendall's Tau) VI
- Correlation Coefficient (Spearman) VI
- Covariance Matrix VI
- Measures of Mean VI
- Measures of Spread VI
- Percentiles VI

The **Trigonometric Functions** subpalette contains the following new functions:

- Inverse Cosecant function
- Inverse Cotangent function
- Inverse Secant function

Signal Processing Palette

The **Filters** subpalette contains the new Zero Phase Filter VI.

The **Signal Operation** subpalette contains the new Riffle VI.

The **Transforms** subpalette contains the following new VIs:

- Chirp Z Transform VI
- DCT VI
- DST VI
- Inverse DCT VI
- Inverse DST VI

The **Waveform Conditioning** subpalette contains the new Continuous Convolution (FIR) VI.

The **Waveform Measurements** subpalette contains the new Extract Multiple Tone Information VI.

The **Windows** subpalette contains the following new VIs:

- Blackman-Nuttall Window VI
- Chebyshev Window VI
- Gaussian Window VI
- Symmetric Window VI
- Window Properties VI

Programming Palette

The **Comparison** subpalette contains the following new functions:

- Empty Array? function
- Polar to Re/Im function
- Re/Im to Polar function

Digital Waveform VIs Enhancements

Use the Digital Waveform VIs to replace a subset of a digital waveform (DWDT) or digital table (DTbl), to convert from digital waveform data or digital data to an analog waveform or binary data format, or to convert from an analog waveform, binary format, or spreadsheet string to digital waveform data or digital data. Use the Digital Pattern Generator to generate a digital pattern using various fill patterns. Use the Empty Digital Data and Empty Digital Waveform constants to return empty digital waveform data or digital data, which is important for initializing shift registers or for building waveforms.

Improved and Changed VIs

The following Digital Waveform VIs are improved or changed in LabVIEW 8.0.

- The Invert Digital VI has two new inputs. **signal index** specifies the signal at which to begin inverting data. **number of signals** specifies the number of signals to invert.
- The **compare mode** input of the Search for Digital Pattern VI specifies how to handle values of X for the search.
- The Analog to Digital VI is polymorphic and can convert an analog waveform to digital data.
- The Digital to Analog VI is polymorphic and can convert digital data to an analog waveform.

New VIs and Functions

LabVIEW 8.0 introduces the following new Digital Waveform VIs and functions.

- Binary to Digital VI
- Digital Pattern Generator VI
- Digital Ring constant
- Digital to Binary VI
- Empty Digital Data constant
- Empty Digital Waveform constant

- Replace Subset VI
- Spreadsheet String to Digital VI

Disable Structures

Use the Conditional Disable structure to disable sections of code by assigning each subdiagram of the structure an expression composed of symbols and values, both user-defined and LabVIEW-defined, which LabVIEW then evaluates to determine which subdiagram to use.

Note LabVIEW does not compile code on the block diagram that does not execute. For example, LabVIEW does not compile any code that exists within the inactive subdiagrams of the Conditional Disable structure. In addition, LabVIEW does not compile code within a Case structure that has a constant wired to it that would not execute the case with the code.

Use the Diagram Disable structure to define parts of the block diagram that you do not want to compile.

Express VIs Enhancements

Use the Express VIs and functions to build common measurement tasks.

Improved and Changed VIs

The following Express VIs are improved or changed in LabVIEW 8.0:

- The Comparison Express VI has two new inputs. **Items to Compare** allows you to compare data points, time stamps, time between points, number of data points, and signal names of the input signals. **Constant Value to Compare Against** specifies the constant value with which to compare the **Operand 1** input. This VI also has been optimized to return more appropriate results in the case of the first input signal being empty.
- The Read LabVIEW Measurement File and Write LabVIEW Measurement File Express VIs were renamed Read From Measurement File and Write To Measurement File, respectively. These Express VIs include support for binary measurement files (`.tdm`) as well as text-based measurement files (`.lvm`). Select the **Binary (TDM)** option in the configuration dialog box to set the file format to a file.
- The Read From Measurement File, Write To Measurement File, Prompt User for Input, and Report Express VIs change behavior depending on their target. If the current target does not or might not have a host computer connected, the configuration dialog boxes display warnings next to options that are invalid without a host. These

VIs return errors if you configure the VIs to prompt for input and execute the VIs on a target with no user interface, such as the LabVIEW Real-Time Module with no host computer connected.

When you target the Report VI to a non-Windows platform, the configuration dialog box supports only **HTML for Web Page** as a destination.

- The Relay Express VI uses the configuration dialog box value of the **Enable** input if you do not wire a value to the **Enable** block diagram input.
- The **Selector Input** input of the Select Signals Express VI allows you to select a subset of signals from the **Signals** input.
- The Write Data and Write To Measurement File Express VIs were optimized to enforce unique channel names. When a file or channel name is repeated, LabVIEW appends an integer to the end of the name to enforce unique names. For example, if you provide the names `sine`, `sine`, `square`, `square`, and `sine`, LabVIEW updates these names to `sine`, `sine 1`, `square`, `square 1`, and `sine 2`, respectively.

New VIs and Functions

LabVIEW 8.0 introduces the following new Express VIs:

- Acquire Sound Express VI
- Dual Channel Spectral Measurement Express VI
- Play Waveform Express VI
- NI DIAdem Report Express VI

NI DIAdem Report Express VI

Use the NI DIAdem Report Express VI to generate reports directly from LabVIEW data. You can export the report as a PDF or HTML file, print the report, or automatically display the report. The report is completely configured in LabVIEW.

Note You must have DIAdem 9.1 Service Pack 2 or later installed to use the NI DIAdem Report Express VI. Refer to the National Instruments Web site at `ni.com` to download an evaluation version of DIAdem or purchase DIAdem.

File I/O VIs and Functions Enhancements

The **File I/O** palette includes updated VIs and new and updated functions to make reading data from and writing data to files easier, including enhancements to the path input and new datalog functions. The **File I/O** palette also includes new VIs that you can use to work with zip files.

Path Input

File I/O functions that accept path or refnum data as inputs include an input with the phrase `(use dialog)`, such as the **path (use dialog)** input. If you do not wire this input, a dialog box appears prompting users to select a file or directory when the function executes. Each of these functions include a **cancelled** output. If users click the **Cancel** button in the dialog box, **cancelled** returns TRUE and the function returns an error. If you wire a path to a File I/O function, the function opens the file with the minimum permissions needed to perform the operation and eliminates the need for explicitly opening or closing the file.

Datalog Functions

Use the Datalog functions to write to and read from datalog files. The **Datalog** palette includes the following new functions:

- Open/Create/Replace Datalog function
- Get Datalog Position function
- Get Number of Records function
- Read Datalog function
- Set Datalog Position function
- Set Number of Records function
- Write Datalog function

File I/O VIs and Functions

Use the File I/O VIs and functions to open and close files, read from and write to files, create directories and files, retrieve directory information, and write strings, numbers, arrays, and clusters to files. The **File I/O** palette contains the following new functions:

- Open/Create/Replace File function
- Read from Binary File function
- Read from Text File function
- Write to Binary File function
- Write to Text File function

Zip VIs

Use the Zip VIs to create new zip files, add files to zip files, and close zip files.

Miscellaneous Enhancements and Changes

LabVIEW 8.0 introduces the following miscellaneous File I/O VI and function enhancements and changes:

- The Copy function includes **overwrite** and **prompt** inputs and a **cancelled** output.
- The Delete function includes the **entire hierarchy** input, which, if TRUE, deletes the entire hierarchy at or below the **path** specified.
- The File Dialog function is now the File Dialog Express VI. Double-click this VI to display the **Selection Mode** dialog box to configure the types of files or directories users can select in the file dialog box.
- The Close File, Copy, Create Folder, Delete, File/Directory Info, Flush File, and Move functions have new connector pane configurations. The new configurations could affect the positions of wires on block diagrams created in previous versions of LabVIEW.
- You can overwrite existing files and directories when you use the Move and Copy functions.
- The File/Directory Info function has two new outputs. **shortcut** indicates whether the file or directory is a shortcut. **resolved path** returns the path to the shortcut target.
- The String instance of the Read Key VI does not contain the **multibyte encoding (0: None)** parameter from LabVIEW 7.*x* and earlier.

Instrument I/O Assistant Enhancements

The Instrument I/O Assistant includes the following enhancements:

- You can select the character that signifies the last character of the command for the instrument from the **Termination character** pull-down menu in the **Select Instrument** step. You also can select **<custom>** to enter a termination character in the **Termination character** text box.
- You can add an input parameter to an instrument command by selecting the command in the **Enter a command** text box of the **Write** step and clicking the **Add parameter** button. You also can select and right-click the command and select **Add parameter** from the shortcut menu.
- You can split a **Query and Parse** step into the **Write** and **Read and Parse** steps to use parameterized inputs. To split a **Query and Parse** step, right-click the step in the **Step Sequence** window and select **Split step into Write and Read steps** from the shortcut menu.
- You can configure the Instrument I/O Assistant to parse all data in a token until you have no data left to parse. To configure the Instrument I/O Assistant to parse until the end of the data, select the **To end of**

data item from the **Count** pull-down menu in the **Query and Parse** step, or click the **To end of data** button below the **Count** pull-down menu. The Instrument I/O Assistant also parses to the end of the data if you click the **Auto parse** button. The **To end of data** item and behavior are available only if the token you configure is the last token.

Sound VIs Enhancements

The **Sound** palette includes the VIs you use to incorporate sound into your VIs and record sound with VIs.

Note **(Windows)** You must have DirectX 8.0 or later to use the Sound VIs.

These VIs include the following enhancements:

- The VIs support up to 24-bit sound.
- The VIs support multi-channel sound files, as well as monophonic and stereophonic sound.
- A waveform represents sound data. You can use elements of unsigned 8-bit, signed 16-bit, or signed 32-bit integers, or single and double-precision data types to represent the Y array data. Each waveform defines one channel.
- The format of the sound data is Pulse Code Modulated (PCM).
- The VIs can produce continuous sound output.
- The VIs allow for a streaming view of wave files.
- The VIs have improvements to error checking.

String Functions Enhancements

Use the Match Regular Expression and the Search and Replace String functions for more complex regular pattern matching using standard Perl Compatible Regular Expression (PCRE) syntax. Right-click the Search and Replace String function and select **Regular Expression** from the shortcut menu to configure the function for advanced regular expression searches and submatch substitution in the replacement string. Resize the Match Regular Expression function to view any submatches found in the string. You can use the Match Pattern function or the Match Regular Expression function to search strings for regular expressions. The Match Regular Expression function gives you more options for matching strings but performs more slowly than the Match Pattern function.

VI Hierarchy Window Enhancements

Select **View»VI Hierarchy** to display the **VI Hierarchy** window, which displays open LabVIEW projects and targets, as well as the calling hierarchy for all VIs in memory. The **VI Hierarchy** window displays a top-level icon to represent the main LabVIEW application instance, under which appear all open VIs that are not in a project. The **VI Hierarchy** window displays a top-level icon to represent each project in memory. Each target you add appears under the project.

Front Panel Enhancements

LabVIEW 8.0 introduces the following front panel enhancements and changes.

Changes to Graph Cursors

LabVIEW 8.0 includes the following changes to graph cursors:

- To create a cursor, right-click anywhere in the cursor legend, select **Create Cursor**, and select a cursor mode from the shortcut menu. The cursor includes the following modes.

Table 4. LabVIEW 8.0 Cursor Mode Changes

LabVIEW 7.1 Cursor Mode	LabVIEW 8.0 Cursor Mode	Description of LabVIEW 8.0 Cursor Mode
Free	**Free**	Moves the cursor freely within the plot area, regardless of plot positions.
Snap to Point and **Lock to Plot**	**Single-Plot**	Positions the cursor only on the plot that is associated with the cursor. You can move the cursor along the associated plot. Right-click the cursor legend row and select **Snap To** from the shortcut menu to associate one or all plots with the cursor.
—	**Multi-Plot**	Positions the cursor only on a specific data point in the plot area. The multi-plot cursor reports values at the specified x value for all the plots that the cursor is associated with. You can position the cursor on any plot in the plot area. Right-click the cursor legend row and select **Snap To** from the shortcut menu to associate one or all plots with the cursor. This mode is valid only for mixed signal graphs.

- You cannot change the mode of a cursor after you create it. You must delete the cursor and create another cursor. Right-click the cursor legend row and select **Delete Cursor** from the shortcut menu to delete a cursor.
- The **Cursor Movement Selector**, **Formatting Ring**, and **Lock Ring** buttons do not appear on the right side of the cursor legend. Right-click a cursor legend row and select options from the shortcut menu to customize the cursor. All the options you accessed using the cursor legend buttons in previous versions of LabVIEW now are in the shortcut menu.
- Use the X Scale property to set the x-scale of the cursor programmatically. This property is valid only if the cursor mode is **Free**. This property is similar to the **X Scale** item in the shortcut menu of the cursor legend of a graph.
- Use the Y Scale property to set the y-scale of the cursor programmatically. This property is valid only if the cursor mode is **Free**. This property is similar to the **Y Scale** item in the shortcut menu of the cursor legend of a graph.
- Use the Watch Plots and Watch All Plots properties to watch plots in the plot area programmatically. These properties are valid only for cursors associated with multiple plots.

Using Graph Annotations

Use annotations on a graph to highlight data points in the plot area. The annotation includes a label and an arrow that identifies the annotation and data point. You also can configure the appearance of an annotation and how the annotation snaps to plots in the plot area. To create an annotation, right-click the graph and select **Data Operations»Create Annotation** from the shortcut menu to display the **Create Annotation** dialog box. Right-click the annotation and select options from the shortcut menu to configure the annotation. Use the Annotation List property to get an array of information about all annotations in the plot area programmatically.

Drawing Images in the Graph Plot Area

You can draw images to the background, foreground, or middle plot area of a graph. Use the Plot Images:Back, Plot Images:Front, and Plot Images:Middle properties for a waveform graph, intensity graph, or mixed signal graph to set the plot area background, foreground, and middle plot image, respectively. You can use the Plot Image:Front property only for an intensity graph. The plot data lies between the foreground and the middle plot image. The grid lines lie between the middle plot image and the background. If you want to remove an existing image from the background, foreground, or middle plot area, you must wire an empty image to the appropriate property or make the image transparent.

Note If you resize a graph, the plot area images do not adjust accordingly. You must rerun the VI to redraw the images.

You must map the graph coordinates to the coordinates of the front panel to draw an image appropriately in the graph plot area. Use the Map XY to Coordinates method to map the graph or chart coordinates to the front panel coordinates. This method adjusts only the left and top plot bounds to orient the plot around the origin of the front panel.

Refer to the `examples\general\graph\Graph Pictures.llb` for examples of using the Plot Image properties and the Map XY to Coordinates method.

Exporting Images of Graphs, Charts, and Tables

You can export images of graphs, charts, tables, digital data, and digital waveform controls and indicators into the following formats:

(Windows) `.emf`, `.bmp`, and `.eps` files

(Mac OS) `.pict`, `.bmp`, and `.eps` files

(Linux) `.bmp` and `.eps` files

To export an image, right-click the control and select **Data Operations» Export Simplified Image** from the shortcut menu. In the **Export Simplified Image** dialog box, select the image format and whether you want to save the image to the clipboard or to disk. You also can use the Export Image method to export an image programmatically.

Displaying Planes on XY Graphs

You can display Nyquist planes, Nichols planes, S planes, and Z planes on the XY graph. To display these planes, right-click the XY graph and select **Optional Plane** from the shortcut menu. The **Optional Plane** menu item contains the various types of planes—Nyquist, Nichols, S, and Z planes—you can display on the XY graph. When you select an optional plane to display, the **Optional Plane** menu contains menu items to show and hide the plane lines, plane labels, and Cartesian lines.

You also can use the Optional Plane:Plane Type property to display these planes on the XY graph programmatically. Use the Optional Plane:Lines Visible and Optional Plane:Labels Visible properties to show and hide plane lines and plane labels on the XY graph programmatically. Use the Cartesian Lines:Visible property to show and hide the Cartesian lines on the XY graph programmatically.

The XY graph also accepts an array of complex data, in which the real part is plotted on the x-axis and the imaginary part is plotted on the y-axis.

Mixed Signal Graph

Use the mixed signal graph to plot data of different types that share a common x-scale. The mixed signal graph accepts all of the data types accepted by the waveform graph, XY graph, and digital waveform graph. In addition, it accepts any cluster that contains any combination of these data types.

When you add multiple plot areas to a mixed signal graph, each plot area has its own y-scale. The mixed signal graph automatically creates plot areas when necessary to accommodate combinations of analog and digital data.

The plot legend on the mixed signal graph is comprised of tree controls and is displayed to the left of the graph plot areas. Each tree control represents one plot area. You can use the plot legend to move plots between plot areas.

You can use the multi-plot cursor on a mixed signal graph to report the values of multiple plots at a particular x-value.

Key Navigation

You can display the **Key Navigation** page of the **Properties** dialog box by right-clicking a control on the front panel and selecting **Advanced»Key Navigation** from the shortcut menu. You also can display this page by right-clicking a control, selecting **Properties** from the shortcut menu, and clicking the **Key Navigation** tab. The **Key Navigation** page includes the following changes:

- The **Key Assignment** section was renamed **Focus**. The **Focus** list now includes these shortcut keys: <Delete>, <Insert>, <Mute>, <Play>, <Volume Up>, <Volume Down>, F13, F14, F15, F16, F17, F18, F19, F20, F21, F22, F23, and F24.
- The **Current Assignments** section was renamed **Existing Bindings**.
- The **Key Navigation** page has a **Toggle** section, which assigns shortcut keys to toggle Boolean controls. You also can use the Toggle Key Binding property to assign a shortcut key to toggle a boolean control.
- The **Key Navigation** page has **Increment** and **Decrement** sections, which assign shortcut keys to increase and decrease the values of Numeric, Slide, Knob, Ring, and Enum controls. You also can use the Increment Key Binding and Decrement Key Binding properties to assign a shortcut key to increase or decrease the value of a numeric control, respectively.

The Key Navigation property was renamed Focus Key Binding. For Boolean controls, this property sets only the focus keyboard shortcut, and the Toggle Key Binding property sets the toggle keyboard shortcut.

Minimum Front Panel Size

In LabVIEW 7.*x* and earlier, the minimum size of the front panel refers to the content area of that pane, not including the scroll bars. In LabVIEW 8.0, for single-pane front panels, the minimum size still refers to the content area of that pane, not including the scroll bars. However, for multi-pane front panels, the minimum size refers to the entire front panel, including any visible scroll bars.

Splitter Bars and Panes

Use splitter bars from the **Containers** palette to split the front panel into two or more panes that you can scroll separately. You can use these panes to design a user interface with several customizable regions. You also can use a splitter bar to designate a pane as a toolbar or a status bar.

Use the Pane properties, methods, and events to control panes programmatically.

Use the Splitter properties and events to control splitter bars programmatically.

In LabVIEW 8.0, the **Show scroll bars** option in the **Customize Window Appearance** dialog box from LabVIEW 7.*x* and earlier now consists of two options, **Show horizontal scroll bar** and **Show vertical scroll bar**. If you add a splitter bar to the front panel, you cannot change these options in the dialog box. The options reflect the settings of the upper-leftmost pane when there are splitter bars on the front panel.

Customizing Run-Time Shortcut Menus

You can customize the run-time shortcut menu for each control you include in a VI. To customize a shortcut menu, right-click a control and select **Advanced»Run-Time Shortcut Menu»Edit** from the shortcut menu to display the **Shortcut Menu Editor** dialog box. Use the dialog box to associate the default shortcut menu or a customized shortcut menu file (`.rtm`) with the control. Use the Shortcut Menu Activation? event and the Insert Menu Items and Delete Menu Items functions to customize shortcut menus programmatically. Refer to the *LabVIEW Help* for more information about customizing run-time shortcut menus for controls programmatically.

You also can add shortcut menus to front panels. To add a shortcut menu to the front panel, use the Shortcut Menu Activation? and Shortcut Menu Selection pane events.

You also can right-click a control and select **Advanced»Run-Time Shortcut Menu»Disable** from the shortcut menu to disable the run-time shortcut menu on a control.

You can disable default shortcut menus for all controls by selecting **File» VI Properties**, selecting **Window Appearance** from the **Category** pull-down menu, clicking the **Customize** button, and removing the checkmark from the **Allow default run-time shortcut menus** checkbox. If you disable default run-time shortcut menus, you still can include custom shortcut menus.

Using XControls

Use XControls to design and create complex controls and indicators in LabVIEW. XControls include the following features:

- You can combine built-in LabVIEW controls and indicators to create an XControl to take advantage of the combined functionality of the LabVIEW controls and indicators.
- When you use an XControl in a VI, the block diagram for that VI is simplified because the XControl includes the behavior of the control.
- Use an XControl library to define the behavior, appearance, abilities, properties, and methods of an XControl.
- You can customize the shortcut menu for an XControl to include new menu items and the existing menu items of the built-in LabVIEW controls or indicators that you used to create the XControl.

In the **Project Explorer** window, right-click **My Computer** and select **New»XControl** from the shortcut menu to create an XControl.

Note You can create and edit XControls only in the LabVIEW Professional Development System. If a VI contains an XControl, you can run the VI in all LabVIEW packages.

Refer to the Simple Dual Mode Thermometer XControl.lvproj in the `labview\examples\general\xcontrols\Dual Mode Thermometer` directory for an example of using XControls.

Using .NET Controls

You can use .NET controls on the front panel as you do ActiveX controls. Create a new control in a .NET container or add .NET controls to the Controls palette for later use. On the block diagram, wire the .NET control to a Property Node, Invoke Node, or Register Event Callback node to set properties, invoke methods, or handle events for the control.

To create a new .NET control, place a .NET container on the front panel. Right-click the .NET container and select **Insert .NET Control** from the shortcut menu to display the **Select .NET Control** dialog box. Select the control you want to create and click the **OK** button.

To add a .NET control to the **Controls** palette, select **Tools».NET & ActiveX»Add .NET Controls to Palette** to display the **Add .NET Controls to Palette** dialog box. Select the .NET control you want to add and click the **OK** button. LabVIEW saves the controls in the `labview\menus\Controls\DotNet & ActiveX` directory by default because all files in this directory automatically appear in the **.NET & ActiveX** palette.

Listboxes, Tree Controls, and Tables

LabVIEW 8.0 introduces extended functionality, including several new properties and methods, for listboxes, tables, digital data controls, and tree controls. These four types of controls share many behaviors and features.

- Right-click a multicolumn listbox or tree control and select **Visible Items»Row Headers** from the shortcut menu to display the row headers. You also can use the Visible Items:Row Headers Visible property of a multicolumn listbox or tree control to show or hide the row headers programmatically.
- Right-click a multicolumn listbox, table, or tree control and select **Visible Items»Vertical Lines** or **Visible Items»Horizontal Lines** from the shortcut menu to show the vertical and horizontal lines. You also can use the Visible Items:Vertical Lines Visible property for a multicolumn listbox, table, or tree control to show the vertical lines programmatically. You can use the Visible Items:Horizontal Lines Visible property for a single-column listbox, multicolumn listbox, table or tree control to show the horizontal lines programmatically.
- Use the Coloring tool to color separately all cells, all row headers, all column headers, or the top left header cell of tables, listboxes, and tree controls.
- If text does not fit in a cell, a tip strip with the full text appears when you run the VI and place the cursor of the Operating tool over the cell.
- Listbox and tree controls have four new item symbols. The appearance of these checkbox symbols depends on the operating system you use. Right-click an item in the control and select **Item Symbol** from the shortcut menu to view all symbols. Listboxes have the same custom-symbol capabilities as the tree control.
- Press the <Shift-Enter> keys to insert an item below the current item in listbox and tree controls. In tree controls, this key combination places the new item at the same level in the hierarchy as the current item, and all items below the new item move down one row.
- Right-click a listbox or tree control and select **Editable Cells** from the shortcut menu to allow users to edit cells at run time. You also can use

the Allow Editing Cells property to programmatically allow users to edit cells in a single-column listbox, multicolumn listbox or tree control. Use the Edit Cell event to handle the event that a user changes text in the cell of a single-column listbox, multicolumn listbox, or tree control.

- Right-click a listbox or tree control and select **Highlight Entire Row** from the shortcut menu to highlight the entire row when you select an item.
- Keyboard selection behavior in listboxes and tree controls matches that of your operating system.

Note If you assign <PageUp>, <PageDown>, <Home>, or <End> as a shortcut key in the **Key Navigation** page of the **.** dialog box for a control, you cannot use that key to select items in listboxes or tree controls.

Listboxes

LabVIEW 8.0 introduces the following enhancements and changes to listboxes:

- Right-click a single-column listbox and select **Visible Items»Horizontal Scrollbar** from the shortcut menu to show the horizontal scroll bar. You also can use the Visible Items:Horizontal Scrollbar Visible property to show or hide the horizontal scroll bar programmatically. The **Visible Items»Scrollbar** item on the shortcut menu of a single-column listbox now is called **Vertical Scrollbar**. The Scrollbar Visible property now is called Visible Items:Vertical Scrollbar Visible.
- Right-click a single-column listbox and select **Visible Items»Column Header** from the shortcut menu to show the column header. You also can use the Visible Items:Column Header Visible property to show or hide the column header programmatically.
- Right-click a row in a single-column listbox and select **Insert Row Before** from the shortcut menu to add a row before the one you selected.
- Right-click a single-column listbox and select **Autosize Row Height** from the shortcut menu to configure the rows to change height to account for font changes and multiple-line text entries. You also can use the Autosizing Row Height property to configure the rows programmatically to change height.
- Right-click a single-column listbox and select **Multi-line Input** from the shortcut menu to allow cell entries to contain multiple lines. You also can use the Multiple Line Input property to programmatically allow cell entries to contain multiple lines.

- Right-click a multicolumn listbox and select **Smooth Scrolling** from the shortcut menu to allow users to scroll the listbox horizontally one pixel at a time. You also can use the Smooth Horizontal Scrolling property to programmatically allow users to scroll the listbox one pixel at a time.

Tree Controls

LabVIEW 8.0 introduces the following enhancements and changes to tree controls:

- You can use the + and – keys on the numeric keypad to open and close items in the tree control. You also can right-click the tree control and select **Open All Items** or **Close All Items** from the shortcut menu to open or close all items in the tree control.
- By default, LabVIEW limits the interactive selection of multiple items in a tree control to items that share the same parent. To allow the selection of items with different parents, right-click the tree control and remove the checkmark from the **Selection Mode»Limit Multiselect to Siblings** item in the shortcut menu. You also can use the Sibling Multiselect property to programmatically limit the selection of multiple items to siblings or to allow the selection of items with different parents.

Note If you enable the **Limit Multiselect to Siblings** shortcut menu item on a tree control when multiple non-sibling items already are selected, subsequent selection behavior might not be intuitive. If you reset the selection to a valid selection, the selection behavior returns to normal.

- Right-click a tree control and remove the checkmark from the **Expand/Contract Symbol Type»Show at Indent Level 0** item in the shortcut menu to hide the expand/contract symbols for items at the highest level in the hierarchy. The indent level 0 items become flush with the left side of the tree control. If you remove the checkmark next to this option, you can still expand or collapse an item at indent level 0 by double-clicking the item. To disable the double-clicking functionality, wire a TRUE constant to the **Discard?** parameter of the Item Close? event. You also can use the Expand/Contract Symbol:Show at Indent Level 0 property to show or hide the expand/contract symbols at indent level 0 programmatically.

Tables

If the horizontal scroll bar of a table is not visible, you cannot scroll to the right using the <Shift-right arrow> keys at run time. If the vertical scroll bar is not visible, you cannot scroll down using the <Shift-down arrow> keys at run time. You can scroll up or left using key navigation even if the scroll bars are not visible.

Dragging and Dropping in String and Tree Controls

String controls and tree controls support built-in dragging and dropping in LabVIEW 8.0. To drag from a string control, use the Operating tool to select text and drag the selection to a control that accepts text, such as another string control or a tree control. To drag from a tree control, use the Operating tool to select an item and drag the item from the tree control to a control that accepts tree data, such as another tree control or a string control.

You can set item dragging and dropping in controls using shortcut menu items or events, properties, and methods. You can use the Get Drag Drop Data function to return drag data from LabVIEW when you perform a drag and drop operation.

Disabling Word Wrapping in String Controls and Indicators

To disable word wrapping in a string control or indicator, right-click the string and remove the checkmark next to the **Enable Wrapping** shortcut menu item. Disabling word wrapping causes the string to wrap at line breaks instead. If you want to disable word wrapping, the string must be in Normal Display mode. After you disable word wrapping, you can display a horizontal scroll bar by right-clicking the string control or indicator and selecting **Visible Items»Horizontal Scrollbar** from the shortcut menu.

You also can use the Enable Wrapping property to disable word wrapping programmatically. You can use the Horizontal Scrollbar Visible property to display the horizontal scroll bar programmatically.

Path Controls

The **Browse Options** page of the **Path Properties** dialog box replaces the **Browse Options** dialog box. Right-click a path control and select **Browse Options** from the shortcut menu to display the **Browse Options** page. This page includes the following enhancements:

- Use the **Button Text** text box to change the button label. You also can use the Browse Options:Button Text property to change the button label programmatically.

- Place a checkmark in the **Treat LLBs as folders** checkbox to allow the user to select a file in an LLB.
- The following options replace the **Selection Mode** pull-down menu: **Files only**, **Folders only**, **Files or folders**, **Existing only**, **New only**, and **New or existing**.

Scroll Bar Controls

Use the horizontal and vertical scroll bar controls to add custom scroll bars to a control with scrollable data. Change the value of a scroll bar by using the Operating tool to click or drag the scroll box to a new position, by clicking the increment and decrement arrows, or by clicking the spaces in between the scroll box and the arrows.

You can use the Doc Min, Doc Max, Increment, Page Size, Coerce Minimum Value, and Coerce Maximum Value properties to specify the page size and minimum and maximum data range limits programmatically.

Array Scroll Bars

You can display vertical and horizontal scroll bars for an array by right-clicking the control or indicator on the front panel and selecting **Visible Items»Vertical Scrollbar** or **Visible Items»Horizontal Scrollbar** from the shortcut menu. Only arrays with more than one dimension can have both a vertical and horizontal scroll bar. Resize a 1D array vertically to add a vertical scroll bar, and horizontally to add a horizontal scroll bar.

You also can use the Vertical Scrollbar Visible property and the Horizontal Scrollbar Visible property to display vertical and horizontal scroll bars programmatically.

Miscellaneous Enhancements and Changes

LabVIEW 8.0 introduces the following miscellaneous front panel enhancements and changes:

- Right-click a graph or chart and select **Advanced»Disable Plot Legend** from the shortcut menu to disable or enable the plot legend at run time. You also can use the Legend:Disable property to disable or enable the plot legend programmatically on a waveform chart, waveform graph, or mixed signal graph. When you disable a plot legend, you cannot customize how the plot appears in the graph or chart at run time.

- In LabVIEW 7.*x* and earlier, the y-scale on the digital waveform graph displays an axis label by default. In LabVIEW 8.0, the y-scale does not display an axis label by default. The default text for the axis label, when you choose to show it, now is **Digital Plots** instead of **Amplitude**.
- The listbox, multicolumn listbox, table, digital data, and tree controls support the mouse wheel for vertical scrolling.
- Right-click a component of a control or indicator you want to customize and select **Advanced»Customize** from the shortcut menu to display a **Control Editor** window that contains only the component of the control or indicator.
- As you move the slider or needle of slide or rotary controls, LabVIEW displays the value of the control in a tip strip. The tip strip uses the same format and precision as the digital display of the control. If the control has more than one digital display, the tip strip uses the format and precision of the digital display that corresponds to the currently active needle or slider. To disable the tip strip, right-click the control, select **Properties** from the shortcut menu, click the **Appearance** tab, and remove the checkmark from the **Show value tip strip** checkbox. Use the Show Value Tip Strip property to disable value tip strips programmatically.
- Numeric controls and indicators accept the following SI prefixes: c, d, da, and h.

LabVIEW MathScript (Windows)

Use the **LabVIEW MathScript Window** to edit and execute mathematical commands, create mathematical scripts, and view numerical and graphical representations of variables. Select **Tools»MathScript Window** to display this window. The **LabVIEW MathScript Window** generates output from and maintains a history of commands that you call, lists variables that you define, and displays variables that you select.

Note You can use MathScript only in the LabVIEW Professional Development System.

You can write functions and scripts for use in the **LabVIEW MathScript Window** or in the MathScript Node.

Use the MathScript Node to evaluate mathematical formulae and expressions on the block diagram. You can save scripts that you create in the **LabVIEW MathScript Window** and load them in the MathScript Node. You also can save scripts that you create in the MathScript Node and load them in the **LabVIEW MathScript Window**.

Matrix Data Type

Use the matrix data type to make modeling of math problems easier and more productive. Existing polymorphic VIs and functions for matrix operations recognize the matrix data type and execute matrix-specific algorithms accordingly. Mathematics VIs and functions used for complex matrix operations accept matrices as inputs and produce matrices for results.

Note In LabVIEW 7.*x* and earlier, the Matrix Exp, Matrix Logarithm, Matrix Power, and Matrix Square Root VIs are available only in the LabVIEW Full and Professional Development Systems. In LabVIEW 8.0, these VIs are available in the Base Package as well.

Use the matrix data type instead of a 2D array to represent matrix data because the matrix data type stores rows or columns of real or complex scalar data for matrix operations, particularly some linear algebra operations. The Mathematics VIs and functions that perform matrix operations accept the matrix data type and return matrix results, which enables subsequent polymorphic VIs and functions in the data flow to perform matrix-specific operations. If a Mathematics VI or function does not perform matrix operations but accepts a matrix data type, the VI or function automatically converts the matrix data type to a 2D array. If you wire a 2D array to a VI or function that performs matrix operations by default, the VI or function automatically converts the 2D array to a real or complex matrix, depending on the data type of the 2D array.

Most Numeric functions support the matrix data type and matrix operations. For example, you can use the Multiply function to multiply a matrix by another matrix or by a number. You can combine basic numeric data types and complex linear algebra functions to create numeric algorithms that perform accurate matrix operations.

In LabVIEW 7.*x* and earlier, you must use the Linear Algebra VIs to perform mathematical operations on matrices. Although a 2D array can store matrix data, LabVIEW treats the array differently from the way it treats matrix data in a matrix control for some key numeric functions, such as some linear algebra functions. LabVIEW 7.*x* and earlier cannot distinguish a 2D array from a matrix, so you must check the dimension size of the array and the array elements before executing the mathematical functions to ensure that the matrix data is valid for a matrix operation. If you use the matrix data type, you do not need to check the dimension size of the array or analyze array elements before executing numeric functions. If the dimensions of the two matrices you wire are different, for example, one is a 2×3 matrix and the other is a 4×4 matrix, the VI or function returns an empty matrix.

Note Coercion dots appear on VIs and functions when the VI or function converts data to or from a matrix or 2D array. This kind of data conversion does not affect performance because LabVIEW stores matrices the same way it stores 2D arrays.

On the block diagram, the matrix data type looks like a real 2D array or complex 2D array data type with a different wire pattern. The VIs and functions that accept the matrix data type automatically support matrix-specific operations when you wire a matrix data type as an input.

When you wire a matrix data type as an input to one of the following functions, a VI that includes subVIs that work with the matrix data type replaces the function:

- Equal? function
- Not Equal? function
- Absolute Value function
- Add function
- Multiply function
- Square Root function
- Subtract function
- Exponential function
- Natural Logarithm function
- Power Of X function
- Re/Im To Complex function
- Complex To Re/Im function
- Polar To Complex function
- Complex To Polar function

The resulting VI has the same icon but contains a matrix-specific algorithm. The node remains a VI if you disconnect the matrix from the input(s). Wire other data types as inputs to restore the original function. If you wire a data type to a function and that data type causes a basic math operation to fail, the function returns an empty matrix or `NaN`. For example, if you wire a matrix with a dimension of 2×3 to one input of the Multiply function, then wire a 4×4 matrix to the other input, the function returns an empty matrix.

Note When you load a VI from LabVIEW 7.*x* in LabVIEW 8.0 and the VI contains Mathematics VIs wired to functions that can use the matrix data type, LabVIEW disables the type definitions and keeps the behavior from LabVIEW 7.*x*. LabVIEW places a red 7.*x* glyph on the function to indicate that the function uses behavior from LabVIEW 7.*x*.

In LabVIEW 7.0 and earlier, only 1D arrays are aligned in memory, which aids performance for array operations. In LabVIEW 7.1 and later, 1D and 2D arrays are aligned in memory. This aids in performance for linear algebra operations and operations involving the matrix data type.

Options Dialog Box Enhancements

Select **Tools»Options** to display the **Options** dialog box. You can browse the pages in the **Options** dialog box by selecting from the **Category** list on the left side of the dialog box. The **Options** dialog box includes the following enhancements:

- The **Miscellaneous** page was renamed the **Environment** page. The **Maximum undo steps per VI** option moved from the **Block Diagram** page to the **Environment** page.
- The **VI Server: TCP/IP Access** page was renamed the **VI Server: Machine Access** page and has renamed options.
- Use the **VI Server: User Access** page to control user access to VIs through the VI Server.
- Use the **Menu Shortcuts** page to set keyboard shortcuts for LabVIEW menu items.
- Use the **Source Control** page, available only in the LabVIEW Professional Development System, to configure source control for a third-party source control provider and set source control options in LabVIEW.
- Use the **Security** page to set security options for LabVIEW.
- Use the **Shared Variable Engine** page to set time synchronization for shared variables.
- Use the **Labels snap to preset positions on controls/terminals**, **Labels locked by default**, and **Default label position** options on the **Front Panel** and **Block Diagram** pages to snap, lock, and position labels and captions for new objects.
- Select the **Tree** option from the **Format** pull-down menu on the **Controls/Functions Palettes** page to display palette items in a tree control. Place a checkmark in the **Sort palette items** checkbox to sort items on the palette in alphabetical order if you selected the **Text** or the **Tree** option from the **Format** pull-down menu.
- Use the **Service name** option on the **VI Server: Configuration** page to set the TCP/IP service name at which the VI Server listens for requests.

Save As and Save for Previous Version Dialog Boxes

Select **File»Save As** on a previously saved VI to display the **Save As** dialog box. Select **File»Save for Previous Version** to display the **Save for Previous Version** dialog box.

The **Save with Options** dialog box was removed. Use the **Save As** dialog box or the **Save for Previous Version** dialog box instead. You also can access the **Source Distribution Properties** dialog box from the **Save As** dialog box to create a source distribution and configure settings for specified VIs to add passwords, remove block diagrams, or apply other settings.

File Size Improvements

Saving VIs from earlier versions of LabVIEW in LabVIEW 8.0 significantly decreases their file size. The file size of VIs you save in LabVIEW 8.0 is about 55% less than the file size of the same VIs in LabVIEW 7.1. The file size of LLBs in LabVIEW 8.0 is about 20% less than the file size of the same LLBs in LabVIEW 7.1.

Using Shared Libraries in Multiple Versions of LabVIEW

In LabVIEW 7.*x* and earlier, if you build a shared library (DLL) in one version of LabVIEW and use that shared library in another version of LabVIEW, LabVIEW might return an error because a different version of the LabVIEW Run-Time Engine calls the shared library. In LabVIEW 8.0, you can use shared libraries in multiple versions of LabVIEW, or in multiple versions of the Run-Time Engine, if you create shared libraries using the `labviewv.lib` instead of the `labview.lib`.

Setting the Window Run-Time Position

Select **File»VI Properties** and select **Window Run-Time Position** from the **Category** pull-down menu to customize the run-time front panel window position and size. You also can use the Front Panel:Run-Time Position methods to customize the run-time front panel window position and size programmatically.

The **Maximized** selection in the **Position** pull-down menu replaces the **Size the front panel to the width and height of the entire screen** checkbox on the **Window Size** page of the **VI Properties** dialog box in previous versions of LabVIEW.

Similarly, the **Centered** selection in the **Position** pull-down menu replaces the **Auto-Center** checkbox in the **Customize Window Appearance** dialog box in previous versions of LabVIEW.

In previous versions of LabVIEW, if you set **Auto-Center** or **Size the front panel to the width and height of the entire screen** for a VI, the VI did not return to the edit mode position when stopped. The VI remained either centered or the size of the screen. LabVIEW 8.0 returns the VI to the edit mode position when stopped.

Setting Run-Time Language Preferences

You can set the language preferences for a stand-alone application in the following ways. The methods listed first take higher precedence than those listed later.

- Run the application with the `-lang` *`language`* command line option, where *`language`* is the English name of the language you want to use.
- Add the following line to the `.ini` file for the application: `AppLanguage=`*`language`*, where *`language`* is the English name of the language you want to use.
- **(Windows)** If you do not specify the language you want to use in the command line or the `.ini` file of the application, the application uses the same language as the operating system.
- If the application does not support the operating system language, the application uses the default language for the application.
- If you do not specify any of the above language preferences, the application uses the first available supported language in alphabetical order. If the application cannot use any of the supported languages, LabVIEW cannot load the application.

VI Server Enhancements

The VI Server provides more detailed error reports than in LabVIEW 7.*x* and earlier. For example, if an error occurs when you use the Open VI Reference function, the error report states which VI is the source of error. If you load a VI without a front panel, you receive an error stating that LabVIEW cannot load the front panel. Error reports also contain the names of specific properties and methods that produce errors.

Using VI Server References

The control reference constant was renamed to the VI Server reference. You can place this reference on the block diagram and link it to the current VI or application or to a control or indicator in the VI. You can use this reference to access the properties and methods for the associated VI, application, control, or indicator.

Use the Operating tool to click the VI Server reference on the block diagram and select **This Application** or **This VI** to link the reference to the current application or VI. You also can right-click the VI Server reference and select **Link to»This Application** or **Link to»This VI** from the shortcut menu. To link the reference to a control or indicator within the VI, click or right-click the reference in the same way and then select the control or indicator to which you want to link from the shortcut menu.

LabVIEW-built executables cannot open VI Server references to polymorphic VIs. When LabVIEW builds an executable, LabVIEW removes the polymorphic VI and instead builds the polymorphic instances into the executable. Perform VI Server operations on the polymorphic instances directly.

NI Security for VI Server

The VI Server includes NI Security, which gives you more control over which clients are allowed to make a VI Server connection. In LabVIEW 7.*x* and earlier, the VI Server allows or denies connections based on the machine address of the client. In LabVIEW 8.0, with NI Security integration, a VI Server can allow or deny connections based on users and groups.

NI Security for VI Server is disabled by default. When NI Security is disabled, you can use only the machine address to allow or deny connections. To enable NI Security, you must specify if a user or group is allowed or denied access. Select **Tools»Options** to display the **Options** dialog box and select **VI Server: User Access** from the **Category** list to display the **VI Server: User Access** page, which you can use to allow or deny access. An empty user control access list means NI Security is disabled for VI Server.

In LabVIEW 8.0, clients open VI Server connections in the same way they do in LabVIEW 7.*x* and earlier. If the server has VI Server user or group restrictions, the client might need to login before making the connection. Use the **NI Security Login** dialog box or login programmatically using the NI Security:Login method. If a user is not allowed to make a connection, the server closes the connection and returns the `kLVE_NISecurity_AuthenticationFailed` (1379) error to the client.

If a client changes while a VI Server connection is open, the client automatically sends a message to the server notifying the server that the user changed. If the server does not allow this new client to connect, the server closes the VI Server connection and returns the `kLVE_NISecurity_AuthenticationFailed` (1379) error.

Class Browser Window

You can use the **Class Browser** window to select available object libraries and view classes, properties, and methods within the selected object library. You can create Property Nodes or Invoke Nodes with the selected property or method. You also can search for classes, properties, and methods or create dotted properties and methods using the **Class Browser** window. Select **View»Class Browser** to display the **Class Browser** window.

You can edit an existing Property or Invoke Node by right-clicking a property on a Property Node and selecting **Select Property** from the shortcut menu or right-clicking an Invoke Node and selecting **Select Method** from the shortcut menu.

Using .NET Assemblies

LabVIEW automatically loads the latest installed version of the .NET Common Language Runtime (CLR). If you develop a .NET application using an earlier version of .NET, install a later version of .NET, and redistribute that application, users with the previous .NET version cannot execute the application. You can use a configuration file to load a specific CLR version. This way, you can install a later version of .NET and continue developing .NET applications, and other users can continue to execute the application with the previous version of .NET.

In LabVIEW 7.*x*, you must select **Tools»Advanced».NET Assembly References** to register private .NET assembly files manually so you can use VIs that refer to those assemblies. In LabVIEW 8.0, you do not need to register private .NET assembly files, and LabVIEW 8.0 does not have the **Tools»Advanced».NET Assembly References** menu item. Instead, you can place a Constructor Node to create a reference to a private .NET assembly that you specify. LabVIEW stores the relative path from the assembly to the VI that contains the .NET object. LabVIEW loads assemblies more efficiently if you store them in the Global Assembly Cache (GAC), in the same directory as the LabVIEW project file, or in a subdirectory of the LabVIEW project.

The .NET CLR uses configuration settings to determine the assembly version to load. These configuration settings might override your request for a specific assembly version. For example, the system administrator can configure your computer to load version 1.0.0.1 instead of 1.0.0.0. If you then try to load version 1.0.0.0, the .NET CLR promotes the assembly version to 1.0.0.1. LabVIEW also launches a dialog box that notifies you of the promotion.

If you move a VI that uses a private assembly to a different folder or computer, you must keep the assembly in the same location relative to the VI. If LabVIEW cannot find an assembly file in the GAC, the directory that contains the project, or the relative path stored in the VI, LabVIEW prompts you to find the assembly file.

If you build a VI that uses a private assembly into a shared library or stand-alone application, LabVIEW copies the associated private .NET assembly files to the `data` subdirectory in the same directory as the library or application.

In LabVIEW 8.0, `.config` files apply to a saved project, shared library, or stand-alone application.

Instrument Driver Finder

The Instrument Driver Finder, available by selecting **Tools»Instrumentation»Find Instrument Drivers**, gives users an easy way to search for and install LabVIEW Plug and Play instrument drivers without having to leave the LabVIEW development environment. The Instrument Driver Finder also displays the instrument drivers in `instr.lib` and searches for connected instruments using NI-VISA.

Note The Instrument Driver Finder is not available on Mac OS.

Instrument Driver VI Wizard

The Instrument Driver VI Wizard extends the existing instrument driver project library. Create or open an instrument driver project, right-click the project library in the **Project Explorer** window, and select **New»Instrument Driver VI** from the shortcut menu to launch the Instrument Driver VI Wizard. Use the wizard to create an instrument driver VI and insert the VI into an instrument driver project library. You also can use the wizard to configure parameters of an instrument command string that correspond to front panel controls and to define how to parse an instrument response.

Instrument Driver Project Wizard

The Instrument Driver Project Wizard, available by selecting **Tools»Instrumentation»Create Instrument Driver Project**, automates the creation of an instrument driver project, including VIs for controlling a programmable instrument such as GPIB, RS-232, Ethernet, and USB instruments. The wizard creates instrument driver VIs and palette menus for viewing the driver VIs in the **Functions** palette. You can create an instrument driver from a class template, a general purpose message-based template, or an existing driver for a similar instrument.

The available class templates are DC Power Supply, Digital Multimeter, Function Generator, and Oscilloscope. You can use the General Purpose template for any message-based instrument. If the instrument is similar to an existing instrument, you can start a new driver based on an existing instrument driver.

If you create a new instrument driver based on an existing instrument driver, the existing instrument driver project must be installed in the `labview/instr.lib` directory before you launch the Instrument Driver Project Wizard. The existing instrument driver must be a project-style driver. You can download and install project-style drivers using the NI Instrument Driver Finder.

The Instrument Driver Project Wizard creates a LabVIEW project that uses a standard format for LabVIEW Plug and Play instrument drivers. The project includes VIs and palette menu files that comply with the instrument driver guidelines. Once the wizard creates the project, you need to customize each VI to work with an instrument. You can use the Instrument Driver VI Wizard to create an instrument driver VI and insert the VI into the instrument driver project library.

Using and Debugging Reentrant VIs

Use LabVIEW to accomplish the following tasks with reentrant VIs:

- You can debug reentrant VIs. You can view an instance of the block diagram of any reentrant VI for debugging purposes; however, you cannot edit the block diagram instance. Within the block diagram, you can set breakpoints, use probes, enable execution highlighting, and single-step through execution. To allow debugging on a reentrant VI, select **File»VI Properties** to display the **VI Properties** dialog box, select **Execution** from the pull-down menu, and place a checkmark in the **Allow debugging** checkbox.
- You can use the **VI Properties** dialog box to set a reentrant VI to open the front panel during execution and optionally reclose it after the reentrant VI runs.
- You can use the VI Server to programmatically control the front panel controls and indicators on a reentrant VI at run time; however, you cannot edit the controls and indicators at run time. You can use many VI Server methods and properties, such as Control Value:Get [Flattened], Control Value:Set [Flattened], and Run VI to communicate with the reentrant instance of a VI, when the reference is to that instance.
- You also can use the VI Server to create a copy of the front panel of a reentrant VI at run time. To copy the front panel of a reentrant VI, use the Open VI Reference function to open a VI Server reference. Wire a strictly typed VI reference to the **type specifier** input or wire **open for**

reentrant run to the **option** input. When you open a reference, LabVIEW creates a copy of the VI. You also can use the VI Server or the **VI Properties** dialog box to open the front panel of the reentrant VI.

- You can configure an Event structure case to handle events for front panel objects of a reentrant VI.
- You can view and control the front panels of reentrant VIs remotely either from within LabVIEW or from within a Web browser, by connecting to the LabVIEW built-in Web Server. When you use the Web Server, multiple users can access a controller interface to a VI. Each connection gets its own instance.
- The Facade VI of an XControl must be a reentrant VI.
- The front panel of a reentrant VI also can be a subpanel.
- In previous versions of LabVIEW, only one front panel appears for every instance of a reentrant VI, so the values of controls and indicators are not always accurate. In LabVIEW 8.0, each instance of a reentrant VI has its own front panel, ensuring accurate values of controls and indicators every time.

When you open a reentrant subVI from the block diagram, LabVIEW opens a clone of the VI instead of the source VI. The title bar of the VI contains `(clone)` to indicate that it is a clone of the source VI. You cannot edit the clone VI.

Creating Probes after a VI Runs

You can configure a VI to retain wire values so that when you create a probe on the block diagram, the probe displays the data that flowed through the wire at the last VI execution. Click the **Retain Wire Values** button to save the wire values at each point in the flow of execution so that when you place a probe on a wire, you can immediately obtain the most recent value of the data that passed through the wire. You must successfully run the VI at least once before you can collect data from any wires using probes. Using probes after a VI runs is useful when the block diagram is complex and you want to debug the VI.

Click the **Retain Wire Values** button on the block diagram toolbar to retain wire values for use with probes.

Click the **Do Not Retain Wire Values** button at any time to refrain from saving wire values at each execution.

Note Choose the **Do Not Retain Wire Values** option to reduce memory requirements and improve performance slightly.

Profiling VIs in Multiple Targets

You can use the **Profile Performance and Memory** window to profile VIs in different targets simultaneously. The **Profile Performance and Memory** window includes a new **Project Library** column that displays the LabVIEW project library (`*.lvlib`) and targets. You can use this column to differentiate VIs that have the same name but are included in different project libraries or loaded on different targets. The **Profile Performance and Memory** window has a new **Select Application Instances** button, which launches the **Select Application Instances** dialog box. Use this dialog box to select the targets for which the **Profile Performance and Memory** window displays data. Next to the **Select Application Instances** button is a legend that shows each target accessed by the **Profile Performance and Memory** window and the color LabVIEW uses to represent it. Select **Tools»Profile»Performance and Memory** to launch the **Profile Performance and Memory** window, which was previously accessible by selecting **Tools»Advanced»Profile VIs**.

Note A LabVIEW project (`*.lvproj`) is a collection of VIs, build specifications, and so on. All files related to a project appear in the **Project Explorer** window. You can profile only one project at a time. If the **Profile Performance and Memory** window is already open, the **Tools»Profile»Performance and Memory** option is disabled.

64-Bit Integer Data Types

Use the 64-bit signed and unsigned integer data types to use, view, and store large integers.

All functions with polymorphic numeric inputs support the 64-bit integer data types. LabVIEW coerces 64-bit inputs for the rotation amount, shift amount, and power of the Rotate, Logical Shift, and Scale By Power Of 2 functions, respectively, to 32-bit integers. You must wire a 64-bit integer to the **string** input of the Decimal String To Number, Hexadecimal String To Number, Octal String To Number, and Fract/Exp String To Number functions if you want the functions to return a 64-bit integer output.

Use the To Quad Integer and To Unsigned Quad Integer functions to convert numbers to 64-bit signed or unsigned integers, respectively.

NI Spy

NI Spy is a tool for monitoring instrument I/O communications while your application runs. You can use NI Spy to capture instrument I/O calls and their results while LabVIEW VIs run. Capturing calls and their results can help you debug problems with instrument communications. **(Windows)** Select **Start»National Instruments»NI Spy** to launch NI Spy. **(Mac OS)** Select **Applications»National Instruments»NI Spy** and double-click the

NI Spy icon to launch NI Spy. **(Linux)** Type `nispy` in the command line to launch NI Spy.

Find and Replace Functionality

In LabVIEW 7.*x* and earlier, you can replace an object on the block diagram by right-clicking the object and selecting **Replace** from the shortcut menu. In LabVIEW 8.0, you can replace multiple objects or pieces of text at once. Use the **Find** dialog box to search VIs for VIs, objects, and text. Then use the **Search Results** window to replace all or only selected items with other objects or text.

Customizable Keyboard Shortcuts

Use the **Menu Shortcuts** page of the **Options** dialog box to set keyboard shortcuts for VI menu items. You can use the <F1> to <F24> function keys without using the <Ctrl> key as a modifier key.

Print Dialog Box Enhancements

The **Print** dialog box includes the following enhancements:

- Use the **Select VI(s)** page to select whether you want to print the current VI or multiple VIs.
- Use the **Complete front panel**, **Visible portion of front panel**, and **VI documentation** options on the **Print Contents** page to select the print content. The **Using the panel**, **Using as a SubVI**, **Complete documentation**, and **Custom** documentation options moved from the **Print Contents** page to the **VI Documentation** page, which you access by selecting the **VI documentation** option.
- The **Custom details** page was renamed to the **VI Documentation** page. Use the **VI Documentation Style** option to set the components in the **VI Documentation** page. The **Icon and connector pane** and **Description** options replaced the **Icon and description** option and its components. Use the **Label**, **Caption**, and **Caption [Label]** components under the **Controls** option to print labels and/or captions for controls and indicators.
- This dialog box prints documentation for VIs in the current application instance. To print documentation for VIs in multiple application instances, you must repeat a print operation in each application instance.

Editing VI Icons

Use the lv_icon VI template in the `labview\resource\plugins` directory to create a VI with which you can edit VI icons outside of the **Icon Editor** dialog box. Refer to the SAMPLE_lv_icon VI for an example of creating an icon editor VI.

Documentation Enhancements and Changes

LabVIEW 8.0 includes the following documentation enhancements and changes.

Documentation Reorganization

The *LabVIEW Help* now contains conceptual content from many of the LabVIEW 7.*x* and earlier manuals and application notes. LabVIEW therefore does not ship with a PDF library or with PDF versions of the *LabVIEW Measurements Manual*, *Using External Code in LabVIEW*, *LabVIEW Development Guidelines*, *LabVIEW Analysis Concepts*, or application notes.

The following table compares the location for documentation in LabVIEW 7.*x* with the location in LabVIEW 8.0.

Table 5. LabVIEW 8.0 Documentation Reorganization

Documentation	LabVIEW 7.*x*	LabVIEW 8.0
Getting Started with LabVIEW	print and PDF	print and PDF
LabVIEW Release Notes	print and PDF	print and PDF
LabVIEW Upgrade Notes	print and PDF	print and PDF
LabVIEW Quick Reference Card	print and PDF	print and PDF
LabVIEW User Manual	print and PDF	*LabVIEW Help*: **Fundamentals** (subset in print)
LabVIEW Measurements Manual	print and PDF	*LabVIEW Help*: **Taking Measurements**; **Controlling Instruments**
LabVIEW Application Builder Readme	print and PDF (*LabVIEW Application Builder User Guide*)	`labview\readme` directory
LabVIEW Development Guidelines	print and PDF	*LabVIEW Help*: **Fundamentals»Development Guidelines**

Table 5. LabVIEW 8.0 Documentation Reorganization (Continued)

Documentation	LabVIEW 7.*x*	LabVIEW 8.0
LabVIEW Analysis Concepts	PDF	*LabVIEW Help*: **Fundamentals»Signal Processing and Analysis**
Using External Code in LabVIEW	PDF	*LabVIEW Help*: **Fundamentals»Calling Code Written in Text-Based Programming Languages**
Integrating the Internet into Your Measurement System—DataSocket Technical Overview	PDF	—
LabVIEW and Hyper-Threading	PDF	*LabVIEW Help*: **Fundamentals»LabVIEW and Hyperthreading**
LabVIEW Custom Controls, Indicators, and Type Definitions	PDF	*LabVIEW Help*: **Fundamentals»Building the Front Panel»Concepts»Creating Custom Controls, Indicators, and Type Definitions**
LabVIEW Data Storage	PDF	*LabVIEW Help*: **How LabVIEW Stores Data in Memory**
LabVIEW Performance and Memory Management	PDF	*LabVIEW Help*: **Fundamentals»Managing Performance and Memory»Concepts»Using the VI Profile Window to Monitor VI Performance**
Polymorphic Units in LabVIEW	PDF	*LabVIEW Help*: **Fundamentals»Creating VIs and SubVIs»Concepts»Using Polymorphic Units**
Porting and Localizing LabVIEW VIs	PDF	*LabVIEW Help*: **Fundamentals»Porting and Localizing VIs**
Using Apple Events and the PPC Toolbox to Communicate with LabVIEW Applications on the Macintosh	PDF	—

Table 5. LabVIEW 8.0 Documentation Reorganization (Continued)

Documentation	LabVIEW 7.*x*	LabVIEW 8.0
Using LabVIEW to Create Multithreaded VIs for Maximum Performance and Reliability	PDF	*LabVIEW Help*: **Fundamentals»Definitions: Multitasking, Multithreading, and Multiprocessing**
Using LabVIEW with TCP/IP and UDP	PDF	*LabVIEW Help*: **Using LabVIEW with TCP/IP and UDP**
Using LabVIEW with Wireless Devices	PDF	*LabVIEW Help*: **Using LabVIEW with Wireless Devices**
Using the Timed Loop to Write Multirate Applications in LabVIEW	PDF	*LabVIEW Help*: **Fundamentals»Loops and Structures»Timed Structures**

(Windows) Complete the following steps to print an entire book from the **Contents** tab in the *LabVIEW Help*:

1. Right-click the book.
2. Select **Print** from the shortcut menu to display the **Print Topics** dialog box.
3. Select the **Print the selected heading and all subtopics** option.

Note Select **Print the selected topic** if you want to print the single topic you have selected in the **Contents** tab.

4. Click the **OK** button.

This help file may contain links to PDF documents. To print PDF documents, click the print button located on the Adobe Acrobat Viewer toolbar.

Use the `Adobe Reader with Search and Accessibility 6.x or later` to search the PDF versions of the *LabVIEW User Manual*, *LabVIEW Release Notes*, *LabVIEW Upgrade Notes*, *Getting Started with LabVIEW Manual*, and *LabVIEW Application Builder Readme*. Complete the following steps to search all the LabVIEW PDF documents that ship with LabVIEW.

1. Select **Edit»Search** to display the **Search PDF** window.
2. Enter a word or phrase in the **What word or phrase would you like to search for** text box.

3. Click the **All PDF Documents in** button and select the `labview\manuals` directory from the drop-down list.
4. Click the **Search** button.

Refer to the Adobe Reader Help for more information about searching all the PDF documents in a folder for a word or phrase.

Readme Files

The `readme` directory in the LabVIEW file structure contains the LabVIEW readme file and readme files for any modules or toolkits you installed. Access the readme files by selecting **Start»All Programs» National Instruments»LabVIEW»Readme**.

Locate Button

In the *LabVIEW Help*, if you navigate to a topic using the **Index**, **Search**, or **Favorites** tabs, click the **Locate** button to find the topic in the table of contents.

Note LabVIEW does not list dialog box reference topics in the table of contents.

Accessing Examples

You can right-click a VI or function on the block diagram or on a pinned palette and select **Examples** from the shortcut menu to open a help topic with links to examples for that VI or function. Click the thumbtack in the upper left corner of a palette to pin the palette.

Context Help Window

In the **Context Help** window, the **Click here for more help** link and the **More Help** button were both renamed to **Detailed help**.

NI Example Finder Enhancements

Use the **Requirements** list on the **Browse** and **Search** tabs of the NI Example Finder to display additional hardware and software that are required to use an example. Place a checkmark in the **Show Requirements list** checkbox on the **General** tab of the **Configure Example Finder** dialog box to display the **Requirements** list in the NI Example Finder.

You can use the NI Example Finder to display or submit comments or ratings for Web examples on the `NI Developer Zone`. The NI Example Finder also includes links to related documents for Web examples. Place a checkmark in the **Show Resources list** checkbox on the **Web** tab of the **Configure Example Finder** dialog box to display the **Resources** list.

Use the **Browse** tab of the NI Example Finder to browse the NI Developer Zone examples that apply to a specific task you want to complete with hardware or software. **(Windows and Linux)** Select the **Task** option in the **Browse according to** section and place a checkmark in the **Include ni.com examples** checkbox to browse Web examples. You can use the **Maximum number of Web query results** field on the **Web** tab of the **Configure Example Finder** dialog box to specify the maximum number of Web examples to display. You also can use the **Web query time limit (seconds)** field on the **Web** tab of the **Configure Example Finder** dialog box to specify how long the NI Example Finder searches for Web examples before it times out.

If you are browsing examples by task, the **Toolkits and Modules Not Installed** folder in the **Double-click an example to open it** list includes examples for toolkits and modules that are not currently installed. If you select an example in the **Toolkits and Modules Not Installed** folder, the **Requirements** list includes a link to more information about the required toolkit or module.

Use the **Search** tab of the NI Example Finder to search for examples installed on your computer by keyword or description. Select the **Keywords** option from the pull-down menu beside the **Search** button to search by keyword. Select the **Descriptions** option from the pull-down menu beside the **Search** button to search the example descriptions. You must remove the checkmark in the **Include ni.com examples** checkbox to search for examples by description. If you searched the examples by description, the text you searched for is highlighted in the **Information** text box.

Use the NI Example Finder to search or browse for VI-based or project-based examples for LabVIEW. When you open a project-based example, the LabVIEW project file (`.lvproj`) opens in the **Project Explorer** window. Double-click the VI under **My Computer** in the **Project Explorer** window to open the example. The behavior of VI-based examples is unchanged.

In LabVIEW 7.*x* and earlier, the NI Example Finder can find only DAQ boards you have installed. In LabVIEW 8.0, the NI Example Finder finds most hardware you have installed. You still can add uninstalled hardware to search by clicking the **Setup** button to display the **Configure Example Finder** dialog box.

Other LabVIEW 8.0 Features and Changes

LabVIEW 8.0 includes the following miscellaneous features and changes.

Changes to Existing VIs and Functions

LabVIEW 8.0 includes the following changes to existing VIs and functions:

- The **duplicate VISA resource name** output of the VISA Configure Serial Port and VISA Serial Break VIs now is called **VISA resource name out**. The **dup VISA USB Intr Event** output of the VISA Get USB Interrupt Data VI now is called **VISA USB Intr Event out**. The **dup VISA resource name** output of all other VISA VIs and functions now is called **VISA resource name out**.
- The **protocol** input of the VISA Assert Trigger function has two new values—6 (PXI: Reserve) and 7 (PXI: Unreserve).
- The VISA Lock Async VI has two new inputs, **lock type** and **requested key**, and one new output, **access key**. Use these parameters to request an exclusive or shared lock for the current session.
- The **Duration** input of the VISA Serial Break VI specifies the length of the break in milliseconds.
- The **event resource class** input of the VISA Wait on Event function now is called **event class** and is optional instead of required.
- The Get Waveform Components function retrieves the **Y** data array by default.
- When you drop the Compound Arithmetic function from the **Numeric** palette, the default mode is Add. When you drop the Compound Arithmetic function from the **Boolean** palette, the default mode is OR.
- Performance of the To Variant, Variant To Data, Variant To Flattened String, and Set Variant Attribute functions improved. Most VIs that call these functions run faster, but performance is improved most notably for VIs with large data sets.
- The **string** parameter of the Search and Replace String function now is called **input string**. The Search and Replace String function also has five new parameters. The **ignore case?** parameter specifies whether the string search is case sensitive. The **multiline?** parameter specifies whether to match the ^ and $ symbols to the beginning and end of a line, respectively, or to the beginning and end of the whole string, respectively. The **number of replacements** parameter returns the number of times LabVIEW replaced the **search string**. This function also now has **error in** and **error out** parameters.
- The **Transparency Thresh** input of the Read PNG File VI incorporates alpha information for a 32-bit PNG image into the **mask** of the resulting **image data**. LabVIEW treats as opaque any alpha

values equal to or greater than **Transparency Thresh**. LabVIEW treats all other alpha values as fully transparent. The value of **Transparency Thresh** must be between 0 and 255.

- The **compare mode** input of the Search for Digital Pattern VI specifies how to handle values of X for the search. The **mode** and **start index/seconds** inputs of the DWDT Search for Digital Pattern instance of this VI now are called **start value format** and **start**, respectively. The **Index** mode now is called **Samples**.
- The **mode** and **index/seconds** inputs of the waveform instances of the Get Y Value VI now are called **Y position format** and **Y position**, respectively. The **actual index/time value** output of these instances now is called **actual Y position**.
- The **waveform data value** output of the DWDT Get Y Value instance of the Get Y Value VI returns the values of the waveform data.
- The **digital data value** output of the DTbl Get Y Value instance of the Get Y Value VI returns the values of the digital data. The **Digital In** input of this instance now is called **digital data in**. The **Digital Out** and **Digital Value** outputs of this instance now are called **digital data out** and **digital value**, respectively.
- The **Mode**, **Waveform in**, **Start**, and **Duration** inputs of the WDT Get Waveform Subset and DWDT Get Waveform Subset instances of the Get Waveform Subset VI now are called **start/duration format**, **waveform in**, **start**, and **duration**, respectively. The **Index** mode now is called **Samples**. The WDT Get Waveform Subset instance also has a new **start/duration format**, **Absolute Time**. The **Waveform out**, **Actual Start**, and **Actual Duration** outputs of the WDT Get Waveform Subset and DWDT Get Waveform Subset instances of the Get Waveform Subset VI now are called **waveform out**, **actual start**, and **actual duration**, respectively.
- The **Digital**, **Start**, and **Number of Samples** inputs of the DTbl Digital Subset instance of the Get Waveform Subset VI now are called **digital data**, **start**, and **number of samples**, respectively. The **Digital Subset** output of this instance now is called **digital data subset**.
- The Get Y Value, Waveform to XY Pairs, Index Waveform Array, Align Waveform Timestamps, Append Waveforms, Copy Waveform dt, and Scale Delta t VIs have instances for 64-bit signed integers. The Waveform to XY Pairs VI also has an instance for 64-bit unsigned integers.
- The Boolean Array to Digital VI has two new inputs. **compress data** specifies whether to compress the digital output. **sample rate** specifies the frequency in samples per second of the output digital waveform.

- The **response on error** input of the Invert Digital VI now is called **response to invert error**. The Invert Digital VI also has two new inputs. **signal index** specifies the signal at which to begin inverting data. **number of signals** specifies the number of signals to invert, beginning with the signal at **signal index**.
- The **use system alert?** input of the Beep VI specifies whether LabVIEW uses the default system alert and ignores **frequency (Hz)** and **duration (msec)**. This VI does not have the **intensity** input or the **error** output from LabVIEW 7.*x* and earlier.
- If the **Prompt to Replace?** input of the Save Report to File VI is TRUE and **report file path** is a path to an existing file, LabVIEW opens a dialog box to confirm that you want to replace the existing file. Otherwise, LabVIEW overwrites the existing file without warning.
- The default **measurement system for column width** for the Append Table to Report VI is **Default**. If you use this VI to append a table to an HTML report, LabVIEW ignores the **measurement system for column width** input. Instead, this VI multiplies the **column width** input by 100 to determine the maximum column width in pixels.
- The **Include Express VI Configuration Information** input of the Append VI List of SubVIs to Report VI specifies whether the report includes configuration information for any **Express** VIs on the block diagram.
- The space constant has been added to the **Strings** palette. The space constant supplies a one-character space string to the block diagram
- The Flatten To String function has two new inputs: **prepend array or string size?**, which indicates whether LabVIEW includes data size information at the beginning of **data string** when **anything** is an array or string, and **byte order**, which sets the endian order of the resulting flattened string. If you use the default values of these new inputs, this function behaves as before. The function also has a new **error in** input and a new **error out** output.
- The Flatten To String function has a new shortcut menu item, **Convert 7.x Data**, which shows the **type string (7.x only)** output and displays the icon for this function with a red **7.x** on it. The **Expose Typedefs** shortcut menu item is only visible if you wire **type string (7.x only)**. The **type string** output was renamed **type string (7.x only)**, and is only visible if the terminal is already wired from a previous version of LabVIEW or if you right-click the function and select **Convert 7.x Data** from the shortcut menu.
- The Unflatten From String function has two new inputs: **data includes array or string size?**, which indicates whether LabVIEW reads data size information from the beginning of an incoming array or string, and **byte order**, which indicates the endian order of the incoming flattened string. If you use the default values of these new inputs, this function

behaves as before. The function also has a new output, **rest of the binary string**, which contains any leftover bytes that this function did not convert. The **err** output changed to a new **error in** input and a new **error out** output.

- The Date/Time to Seconds function has a new input, **is UTC**, which specifies whether **date time rec** is in Universal Time or in the configured time zone for the computer. In LabVIEW 7.*x*, the Date/Time to Seconds function ignored the **is DST** parameter because the function computed based on Universal Time. If **is UTC** is FALSE (default), LabVIEW 8.0 converts the time based on **is DST**. When you open a VI saved in LabVIEW 7.*x* or previous, LabVIEW assigns the value `-1` to **is DST** to avoid the conversion.
- The Date/Time to Seconds and Seconds to Date/Time functions have a new element in the **date time rec** cluster, **fractional second**, which is the fractions of a second since the start of the second.
- The **to UTC** input of the Seconds to Date/Time function specifies whether **date time rec** is in Universal Time or in the configured time zone for the computer.
- The Format Date/Time String function has a new input, **UTC format**, which specifies whether the output string is in Universal Time or in the configured time zone for the computer.
- The Format Into String and Scan From String functions include support for time stamp data. The **Edit Format String** dialog box has **Format relative time** and **Format time stamp** conversion options, and the **Edit Scan String** dialog box has **Scan relative time** and **Scan time stamp** conversion options.
- Use the `$` format specifier syntax element with the Format Into String function to specify the order of variables in the format string.
- The **Create if not found** input of the String, DBL, SGL, I32, I16, U8, and Time stamp instances of the Set Property VI specifies whether to create a custom property if the property you specify in **Property name** does not exist.

New Example VIs

Refer to the **New Examples for LabVIEW 8.0** folder on the **Browse** tab of the NI Example Finder to view descriptions for and launch example VIs added to LabVIEW 8.0.

Changes to the External Code Functions

You can call the following new functions from a DLL or CIN.

- `CToLStr` converts a C string to a LabVIEW string.
- `LToCStr` converts a LabVIEW string to a C string.

- `DSSetHandleFromPtrNULLMeansEmpty` allocates a handle from a pointer and copies the data from the pointer into the handle. This function is similar to the `DSSetHandleFromPtr` function except that when the pointer has no data, the `DSSetHandleFromPtrNULLMeansEmpty` function nulls out the handle rather than allocating a handle for zero bytes.
- `NIGetOneErrorCode` converts a numeric error code to the appropriate text description.

The CIN does not support the `AZHPurge`, `AZHNoPurge`, `AZHLock`, or `AZHUnlock` functions from LabVIEW 7.*x* and earlier.

Renamed Properties and Methods

LabVIEW 8.0 includes new VI Server classes, properties, methods, and events. Refer to the *LabVIEW Help* for a list of these new VI Server items. The following properties and methods also were renamed in LabVIEW 8.0.

Properties

The following properties were renamed.

Table 6. Renamed Properties in LabVIEW 8.0

Class	LabVIEW 7.1 Name	LabVIEW 8.0 Name
Application	Application:Real-Time Host Connected	Application:User Interface Available
DigitalTable	Cell BG Color	Active Cell:Cell Background Color
DigitalTable	Cell FG Color	Active Cell:Cell Foreground Color
DigitalTable	Top Left Visible Cell	Top Left Cell
Listbox	Scrollbar Visible	Visible Items:Vertical Scrollbar Visible
Listbox	Symbols Visible	Visible Items:Symbols Visible
MulticolumnListbox	Cell Background Color	Active Cell:Cell Background Color
MulticolumnListbox	Cell Size	Active Cell:Cell Size
MulticolumnListbox	Column Headers Visible	Visible Items:Column Headers Visible
MulticolumnListbox	Horizontal Scrollbar Visible	Visible Items:Horizontal Scrollbar Visible
MulticolumnListbox	Symbols Visible	Visible Items:Symbols Visible
MulticolumnListbox	Vertical Scrollbar Visible	Visible Items:Vertical Scrollbar Visible
Scale	Minor Tick Color	Tick Colors:Minor Tick Color

Table 6. Renamed Properties in LabVIEW 8.0 (Continued)

Class	LabVIEW 7.1 Name	LabVIEW 8.0 Name
String	Scrollbar Visible	Vertical Scrollbar Visible
Table	Cell BG Color	Active Cell:Cell Background Color
Table	Cell Size	Active Cell:Cell Size
Table	Column Headers Visible	Visible Items:Column Headers Visible
Table	Horizontal Scrollbar Visible	Visible Items:Horizontal Scrollbar Visible
Table	Index Visible	Visible Items:Index Visible
Table	Row Headers Visible	Visible Items:Row Headers Visible
Table	Vertical Scrollbar Visible	Visible Items:Vertical Scrollbar Visible
TreeControl	Allow Dragging	Drag/Drop:Allow Item Dragging
TreeControl	Allow Dragging Between Items	Drag/Drop:Allow Dropping Between Items
TreeControl	Allow Dragging of Parent Items	Drag/Drop:Allow Dragging of Parent Items
WaveformChart	Autosize Legend	Legend:Autosize
WaveformGraph	Autosize Legend	Legend:Autosize

Methods

The following methods were renamed.

Table 7. Renamed Methods in LabVIEW 8.0

Class	LabVIEW 7.1 Name	LabVIEW 8.0 Name
Control	Reinit To Dflt	Reinitialize To Default
Listbox	Get DblClk Row	Get Double-Clicked Row
MulticolumnListbox	Get DblClk Row	Get Double-Clicked Row
Text	MoveToDefLoc	Move to Default Location
TreeControl	Custom Item Symbols:Revert Symbol	Custom Item Symbols:Revert To Built In Symbol
TreeControl	Custom Item Symbols:Revert Symbols	Custom Item Symbols:Revert All To Built In Symbol

Table 7. Renamed Methods in LabVIEW 8.0 (Continued)

Class	LabVIEW 7.1 Name	LabVIEW 8.0 Name
TreeControl	Custom Item Symbols:Set Symbol	Custom Item Symbols:Set To Custom Symbol
TreeControl	Custom Item Symbols:Set Symbol Array	Custom Item Symbols:Set To Custom Symbol Array
VI	Close FP	Front Panel:Close
VI	Export VI Strings	VI Strings:Export
VI	Get All Control Values	Control Value:Get All [Flattened]
VI	Get All Control Values [Variant]	Control Value:Get All [Variant]
VI	Get Control Value	Control Value:Get [Flattened]
VI	Get Control Value [Variant]	Control Value:Get [Variant]
VI	Get Diagram Image Scaled	Block Diagram:Get Image Scaled
VI	Get Lock State	Lock State:Get
VI	Get Panel Image	Front Panel:Get Panel Image
VI	Get Panel Image Scaled	Front Panel:Get Panel Image Scaled
VI	Get VI Icon as Image Data	VI Icon:Get As Image Data
VI	Import VI Strings	VI Strings:Import
VI	Lock Remote Panel Control	Remote Panel:Lock Control
VI	Make Current Values Default	Default Values:Make Current Default
VI	Open FP	Front Panel:Open
VI	Reinitialize All To Default	Default Values:Reinitialize All To Default
VI	Save VI Icon to File	VI Icon:Save To File
VI	Set Control Value	Control Value:Set [Flattened]
VI	Set Control Value [Variant]	Control Value:Set [Variant]
VI	Set Lock State	Lock State:Set
VI	Set VI Icon from File	VI Icon:Set From File

Table 7. Renamed Methods in LabVIEW 8.0 (Continued)

Class	LabVIEW 7.1 Name	LabVIEW 8.0 Name
VI	Set VI Icon from Image Data	VI Icon:Set From Image Data
VI	Unlock Remote Panel Control	Remote Panel:Unlock Control

Error Code Enhancements

You can use error codes –8999 through –8000 to define custom error messages.

LabVIEW 8.0 introduces the following changes to error codes:

Table 8. LabVIEW 8.0 Error Code Changes

LabVIEW 7.1 Error Code	LabVIEW 8.0 Error Code
20003	20012
20101	20111
20102	20112
20103	20113
20104	20114

Miscellaneous

LabVIEW 8.0 includes the following miscellaneous changes:

- The **Registration Information** dialog box, which previously appeared any time you launched an unregistered version of LabVIEW, does not appear in LabVIEW 8.0. LabVIEW 8.0 collects information such as your name and the name of your company during installation.
- **(Windows)** In LabVIEW 7.*x* and earlier, pressing the <Ctrl-Tab> keys cycles through open LabVIEW windows based on the order in which you opened the windows. In LabVIEW 8.0, pressing the <Ctrl-Tab> keys cycles through open LabVIEW windows based on the order in which the windows appear onscreen. This order is the same as the order you see when you press the <Alt-Tab> keys. **(Linux)** The order of the windows depends on the window manager you use.
- If you drag an item in a tree control under a child-only item, LabVIEW places the item below the child-only item at the same hierarchical level.
- Use the VI Refnum (Panel Image) probe to display the VI name, the VI path, the VI front panel image, and the hex value of a VI reference.

You can use this probe to set a breakpoint if the value is an invalid refnum.

- The Disabled Items property for the listbox and multicolumn listbox returns an error if you set this property for an item that is not in the listbox.
- **(Mac OS)** LabVIEW supports the mouse scroll wheel. You can scroll through the subdiagrams of a Case, Event, and Stacked Sequence structure by moving the cursor over the selector label and pressing the <Command> key while moving the mouse wheel.
- **(Mac OS)** LabVIEW only accepts <Control>-click for popups and will not receive the <Command>-click key combination.
- When you are in customize mode in the **Control Editor** window, you can import a picture from file for a part of a custom control using the **Import from File** item on the shortcut menu. You also can replace a decoration with an image from the clipboard by selecting the decoration and selecting **Edit»Paste**. Right-click the decoration and select **Import from File** from the shortcut menu to replace the decoration with an image from a file.
- You cannot specify a custom **Menus Directory** on the **Paths Options** page of the **Options** dialog box.
- When specifying a custom **VI Search Path** on the **Paths Options** page, you can use the <`osdatadir`> symbolic path.
- When you convert a time stamp to a variant, the variant indicator displays the current value of the time stamp that you wired to it.
- The CVI Function Panel Converter and the **Update VXIPNP Drivers** dialog box are not available in LabVIEW 8.0. Download the LabVIEW Interface Generator for LabWindows™/CVI™ Instrument Drivers tool from `ni.com/idnet` to obtain this functionality.
- The **End text entry with Enter Key** option is on the **Environment** page of the **Options** dialog box.
- The **Explain Changes** dialog box contains a **Path** text box that lists the path on disk of the selected VI. You can use the **Path** text box to distinguish between VIs that have the same name.
- The performance improvement for disabling debugging for a VI increased slightly.
- The **VI Library Manager** window now is called the **LLB Manager** window. Right-click an item in the **Files** list and select **Top Level?** from the shortcut menu to indicate whether an item in an LLB is at the top level. You cannot cancel changes you make in the **LLB Manager** window.

- The **Edit VI Library** dialog box has been removed from the **Tools** menu. Use the **LLB Manager** window to delete a VI from a library or to mark a VI as a top-level VI in a library.
- In LabVIEW 7.*x* and earlier, when you wire an error cluster to a Case structure, the `True` case becomes the `Error` case and the `False` case becomes the `No Error` case. In LabVIEW 8.0, when you wire an error cluster to a Case structure, the `True` case becomes the `No Error` case and the `False` case becomes the `Error` case.
- You can create a specific property or method on the block diagram by right-clicking the control or indicator on the front panel, selecting **Create»Property Node** or **Create»Invoke Node**, and selecting a property or method from the shortcut menu.
- **(Windows)** Use the CAN Channel control located on the **CAN Controls** palette with NI-CAN to access the NI-CAN Channel API. For more information on controls located on the **CAN Controls** palette, refer to the *CAN Initialize* topic in the NI-CAN documentation.
- On the **Format and Precision** page of a **Properties** dialog box, when **Precision Type** is **Significant digits**, National Instruments recommends that you use values from 1 through 6 in the **Digits** field for single-precision, floating-point numbers and values from 1 through 13 in the **Digits** field for double-precision and extended-precision, floating-point numbers.
- On the **Scales** page of the Chart, Intensity Chart, and Intensity Graph **Properties** dialog boxes, place a checkmark in the **Ignore waveform stamp on x-axes** checkbox when you select X-Axis as the scale to set the beginning of the x-scale to 0 instead of the value specified by t0. Place a checkmark in the **Expand digital buses** checkbox when you select Y-Axis as the scale to display digital waveform data as individual data lines.
- You can open multiple files at once. **(Windows)** Select **File»Open** to display a standard file dialog box and press the <Shift> key or the <Control> key to select multiple files. **(Mac OS)** Press the <Shift> key or the <Option> key. **(Linux)** Press the <Shift> key or the <Alt> key.
- All browsers support the **Monitor** option on the **Web Publishing Tool** dialog box.
- In LabVIEW 7.*x* and earlier, when you select **Tools»Compare»Compare VIs** and click the **Select** button, the **Select VI By Name** dialog box appears. In LabVIEW 8.0, when you select **Tools»Compare»Compare VIs** and click the **Select** button, the **Select a VI** dialog box appears.
- In LabVIEW 8.0, when you select **Tools»Compare»Compare VI Hierarchies**, you can select a VI from the file system or from a list of VIs in memory.

- In LabVIEW 7.*x*, you can right-click any block diagram object and select ***Source*** **Palette** from the shortcut menu to access similar objects from a subpalette, where ***Source*** is the name of the subpalette that contains the block diagram object. In LabVIEW 8.0, you also can right-click a block diagram object and select **Replace»*Source* Palette** from the shortcut menu to replace the block diagram object with a similar object from the subpalette that contains the block diagram object. You also can right-click a wire and select **Insert»*Source* Palette** from the shortcut menu to insert an object from the most commonly used source palette. This option inserts the selected object between the objects that the wire connects.
- In LabVIEW 7.*x* and earlier, when you change the data type of a type definition and the instances of the type definition update, the instances preserve only the current data, default data, private data, label, and caption. In LabVIEW 8.0, the instances of the type definitions might preserve more attributes.
- In LabVIEW 8.0, when you replace a type definition with another control, the control might preserve fewer attributes of the type definition than it does in LabVIEW 7.*x* and earlier.
- LabVIEW propagates type definitions whenever possible.
- The **Set Width and Height** dialog box now is called the **Resize Objects** dialog box. In the **Current Size Values** listbox of this dialog box, an asterisk appears next to values for objects that cannot be resized even if you change the **Width** or **Height**.
- In LabVIEW 7.*x*, if LabVIEW cannot find a VI, you must replace the missing VI with a VI of the same name. In LabVIEW 8.0, you can replace a missing VI with a VI of any name. LabVIEW replaces all instances of the missing VI with the VI that you select.
- In LabVIEW 7.*x* and earlier, you can press the <Ctrl-A> keys to repeat an alignment operation for objects on the front panel or block diagram. In LabVIEW 8.0, press the <Ctrl-Shift-A> keys to repeat an alignment operation. Press the <Ctrl-A> keys to select all objects on the front panel or block diagram.
- You can configure when LabVIEW loads a subVI. Right-click a subVI and select **Call Setup** from the shortcut menu to display the **VI Call Configuration** dialog box. This shortcut menu item is available only for VIs open in an application instance that supports VI Server calls—not for functions, Express VIs, polymorphic VIs, or VIs open in an application instance that does not support VI Server calls, such as PDA and FPGA targets. If you have a large caller VI, you can save load time and memory by selecting the **Load and retain on first call** option in the dialog box. When you select this option, the subVI does not load until the caller VI needs it, and you can release the subVI from memory after the operation completes.

- In LabVIEW 8.0, when you copy front panel controls from the block diagram and paste them onto the block diagram of a new VI, the front panel placement will differ from that of the original VI. The front panel objects will be placed in the upper left corner to avoid overlapping or appearing in a non-visible region.
- The LabVIEW display range of years extended from 1904 through 2038 to 1600 through 3000.
- If you want to display a carriage return in the **Context Help** window, you must separate paragraphs with two carriage returns.
- If a key press matches a keyboard shortcut in the VI menu, such as <Ctrl-C> or <Ctrl-V>, LabVIEW does not generate a Key Down event, regardless of whether the menu item is enabled.
- The Application:Language property can return **`ko`** to indicate that the language of the LabVIEW environment is Korean.
- Use the VIs Strings:Export and VIs Strings:Import methods to export multiple VIs to a tagged text file and import multiple VIs from a tagged text file, respectively.

371780A-01 Aug05