LabVIEW™ Upgrade Notes

Upgrading to LabVIEW 8.5

These upgrade notes describe the process of upgrading LabVIEW for Windows, Mac OS, and Linux to version 8.5, issues you might encounter when you upgrade, and new features.

If you are upgrading from LabVIEW 7.1 or earlier to LabVIEW 8.5, National Instruments recommends that you review the following documents in addition to these upgrade notes for more information about enhancements, changes, and features added to LabVIEW between LabVIEW 7.1 and LabVIEW 8.5.

- **LabVIEW 8.0 Upgrade Notes**—The *Upgrade and Compatibility Issues* section and the *LabVIEW 8.0 Features and Changes* section provide important information for upgrade users. Refer to the National Instruments Web site at `ni.com/info` and enter the info code `upnote8` to access the *LabVIEW 8.0 Upgrade Notes*.
- **LabVIEW 8.2 Upgrade Notes**—The *Upgrade and Compatibility Issues* section and the *LabVIEW 8.2 Features and Changes* section provide important information for upgrade users. Refer to the National Instruments Web site at `ni.com/info` and enter the info code `upnote82` to access the *LabVIEW 8.2 Upgrade Notes*.

Refer to the *LabVIEW Help* for more information about LabVIEW 8.5 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, tools, properties, methods, events, dialog boxes, and so on. 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 8.5

If you are upgrading from a previous version of LabVIEW, read this section, *Upgrading to LabVIEW 8.5,* and the *Upgrading from LabVIEW x.x* sections in the *Upgrade and Compatibility Issues* section of this document first, where *x.x* is the version of LabVIEW from which you are upgrading.

Converting VIs

You cannot open a VI saved in a version of LabVIEW prior to LabVIEW 6.0 without contacting an NI representative to obtain conversion software to upgrade your code to VI formats compatible with LabVIEW 8.5. When you open a VI last saved in LabVIEW 6.0 or later, LabVIEW 8.5 automatically converts and compiles the VI. You must save the VI in LabVIEW 8.5, 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.

Note VIs you save in LabVIEW 8.5 do not load in earlier versions of LabVIEW. Select **File»Save for Previous Version** to save VIs so they can run in LabVIEW 8.2 or 8.0. Before saving VIs in LabVIEW 8.5 after you convert them, keep a backup copy of VIs you plan to use in LabVIEW 8.2 or 8.0.

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 Modules, Toolkits, Instrument Drivers, and Add-Ons

After you install LabVIEW 8.5, make sure you have a compatible version of any toolkits and add-ons, then reinstall the toolkits and add-ons in the LabVIEW 8.5 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.

LabVIEW module versions must match the LabVIEW version. For example, you must use the LabVIEW Real-Time Module 8.5 with LabVIEW 8.5. LabVIEW 8.5 might not support add-ons designed for previous versions of LabVIEW.

Refer to the National Instruments Web site at `ni.com/info` and enter the info code `compat` for more information about which LabVIEW modules and toolkits are compatible with the current version of LabVIEW.

The following instrument drivers require upgrades or downloads for use in LabVIEW 8.5:

- 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`.

- The Instr Get Attribute VI and Instr Set Attribute VI no longer ship with LabVIEW. The instrument driver for the Panasonic VP7723A/VP7724A Audio Analyzer used these VIs. To obtain a LabVIEW 8.5 compatible version of the VP7723A/VP7724A driver, download that driver from the National Instruments Instrument Driver Network at `ni.com/idnet`.

You also must mass compile existing toolkit, instrument driver, and add-on VIs for use in LabVIEW 8.5. Because the LabVIEW module version must match the LabVIEW version, you do not have to mass compile the module VIs. Refer to the *Converting VIs* section of this document for more information about mass compiling VIs.

Upgrading Additional National Instruments Software

You must use NI TestStand 3.5 or later in LabVIEW 8.*x*. Refer to the National Instruments Web site at `ni.com/info` and enter the info code `exd8yy` to access the Upgrade Advisor and purchase NI TestStand 3.5 or later.

NI TestStand 3.5 and NI TestStand 4.0 return an error when you attempt to configure the following LabVIEW 8.5 Express VIs:

- Close Data Storage
- Formula
- Get Properties
- Open Data Storage
- Read Data
- Set Properties
- Spectral Measurements
- Write Data

Refer to the National Instruments web site at `ni.com/info` and enter the info code `rdtf10` for more information about the error.

You must use NI Spy 2.3 or later in LabVIEW 8.*x*. NI Spy 2.5 is available on the National Instruments Device Drivers CD.

You must use Measurement Studio 8.0 or later with LabVIEW 8.5. Refer to the National Instruments Web site at `ni.com/info` and enter the info code `exd8yy` to access the Upgrade Advisor and order Measurement Studio 8.0.

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 8.5 without uninstalling previous versions of LabVIEW.

Replacing an Existing Version of LabVIEW

To replace your existing version of LabVIEW, uninstall the existing version of LabVIEW, run the LabVIEW 8.5 installer, and set the 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 8.5 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.

Copying Environment Settings from a Previous Version of LabVIEW

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 8.5 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 8.5, copy the LabVIEW preferences file to the LabVIEW 8.5 directory.

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

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

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

Copying user.lib Files from a Previous Version of LabVIEW

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 8.5, copy the files to the `user.lib` directory in the LabVIEW 8.5 directory.

Upgrade and Compatibility Issues

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

Refer to the `readme.html` file in the `labview` directory for known issues, additional compatibility issues, and information about late addition features in LabVIEW 8.5.

Upgrading from LabVIEW 8.2

You might encounter the following compatibility issues when you upgrade to LabVIEW 8.5 from LabVIEW 8.2.

Note You also can refer to the National Instruments Web site at `ni.com/info` and enter the info code `ex5d8c` for more information about additional issues you may encounter upgrading from LabVIEW 8.2.*x*.

Platforms Supported

LabVIEW 8.5 includes the following changes in platforms supported:

- LabVIEW 8.5 supports Windows Vista x86 and x64.
- LabVIEW 8.5 supports Macintosh computers with both Intel and PowerPC processors.

System Requirements

(Windows) LabVIEW 8.5 requires at least 1.2 GB of disk space for the LabVIEW installation.

(Mac OS) LabVIEW 8.5 requires at least 502 MB of disk space for the LabVIEW minimum LabVIEW installation or 734 MB disk space for the complete LabVIEW installation.

(Linux) LabVIEW 8.5 requires at least 450 MB of disk space for the minimum LabVIEW installation or 640 MB disk space for the complete LabVIEW installation.

Printed Documentation

The following documents did not change for LabVIEW 8.5. Therefore, the content might not reflect changes made in LabVIEW 8.5.

- **(LabVIEW 8.2 and 8.5)** *LabVIEW Quick Reference Card*
- **(LabVIEW 8.2 and 8.5)** *LabVIEW Fundamentals Manual*—Because the *LabVIEW Fundamentals Manual* is a subset of the **Fundamentals** book in the *LabVIEW Help*, refer to the **Fundamentals** book on the **Contents** tab of the *LabVIEW Help* for updated content.

Windows Vista Compatibility Issues

LabVIEW 8.5 supports the Windows Vista OS on 32 and 64-bit systems with the following functionality changes.

The In Port and Out Port VIs do not appear on the **Functions** palette because they allow read/write access to any I/O port on the system, which is discouraged for security reasons on the Vista OS.

- **(Windows Vista x86)** VI components install properly but show up as unsigned in the Windows Defender log. The VIs do run properly.
- **(Windows Vista x64)** These VIs return error -4850.

VI and Function Behavior Changes

The behavior of the following VIs and functions changed in LabVIEW 8.5.

Enhancements to Analysis VIs and Functions

In each version of LabVIEW, National Instruments enhances many of the algorithms behind LabVIEW and C functions. National Instruments also upgrades LabVIEW to use the latest compilers. These enhancements, along with changes in computer hardware and software, might cause differences in the numerical results between LabVIEW 8.2.*x* or earlier and LabVIEW 8.5. When you compare double-precision, floating-point numbers, you might notice small differences on the order of 1E–16. Refer to the National Instruments Web site at `ni.com/info` and enter the info code `exiigr` for more information about comparing floating-point numbers.

Mathematics VIs

LabVIEW 8.5 includes changes to the following **Mathematics** VIs:

- **Find All Zeroes of f(x)**—This VI was renamed to the Find All Zeros of f(x) VI.
- **Zeroes and Extrema of f(x)**—This VI was renamed to the Zeros and Extrema of f(x) VI.

Numeric Functions

LabVIEW 8.5 includes changes to the following **Numeric** functions:

- **Round Toward +Infinity**—The Round To +Infinity function was renamed the Round Toward +Infinity function.
- **Round Toward -Infinity**—The Round To -Infinity function was renamed the Round Toward -Infinity function.

Signal Processing VIs

In the Transition Measurements VI, the **preshoot** output changed to **pre-transition**. This output also changed from a 64-bit double-precision, floating-point numeric data type to a cluster data type. The **overshoot** output changed to **post-transition**. This output also changed from a 64-bit double-precision, floating-point numeric data type to a cluster data type.

Hyperbolic Functions

LabVIEW 8.5 includes changes to the following hyperbolic functions:

- The Inverse Hyperbolic Cosine function returns `NaN` when the input value is a real number that is out of range for the function.
- The Inverse Hyperbolic Secant function returns `NaN` when the input value is a real number that is out of range for the function.

Libraries & Executables VIs and Functions

In the Call Library Function Node, when configuring a Pascal string pointer, you must wire a value to the string input on the block diagram. When configuring a C string pointer, you must wire a value to the string input or specify the string size in the **Minimum size** pull-down menu on the **Parameters** tab of the **Call Library Function** dialog box. You cannot run the VI until you specify values for the strings.

Open VI Reference Function

In LabVIEW 8.0.*x* and 8.2.*x*, if the name of the VI from the **vi path** input matches the name of a VI in memory on that target, LabVIEW returns a reference to the VI in memory and LabVIEW does not load the VI specified in the **vi path** input. In LabVIEW 8.5, if the name of the VI from the **vi path** input matches the name of a VI in memory on that target but the paths differ, the Open VI Reference function returns an error.

Polymorphic VI Terminals that Support 64-bit and Double-Precision Numeric Data Types

LabVIEW coerces extended-precision numeric data to double-precision numeric data if you wire it to a terminal of a polymorphic VI that supports both the double-precision numeric and 64-bit integer types. This coercion preserves a portion of the fractional component of the original data.

Miscellaneous VI and Function Behavior Changes

LabVIEW 8.5 includes the following miscellaneous VI and function behavior changes:

- The Instr Get Attribute VI and Instr Set Attribute VI no longer ship with LabVIEW. If you use either of these VIs in an application, replace them with the Property Node on the **VISA Advanced** palette for equivalent functionality.
- The **All Folders** parameter of the Recursive File List VI can contain folder shortcuts, but the VI does not recurse them.

Property, Method, and Event Behavior Changes

The behavior of the following properties, methods, and events changed in LabVIEW 8.5:

- The Data Binding:Path property of the Control class is read/write and settable when the VI is running. To write this property, you must bind the control to an NI Publish-Subscribe-Protocol URL before you begin writing.
- The Target:Operation System property of the Application class includes the values `Windows x64` and `Linux x64`.
- The Point to Row Column method of the TreeControl class returns the tag `TREE_COLUMN_HEADERS` when you wire a point within the column headers of the tree.
- The LabVIEW Class:Create method includes a name input. If you do not wire the **name** input, LabVIEW prompts the user to name the class at run time.
- The Control Value:Get [Variant], Control Value:Get [Flattened], Control Value:Set [Variant], and Control Value:Set [Flattened] methods no longer trim leading and trailing whitespace when searching for controls.

Deprecated Properties, Methods, and Events

LabVIEW 8.5 does not support the following properties, methods, and events:

- Default Instance property of the LVClassLibrary class. Use the Get LV Class Default Value VI instead.
- Geometry property of the SceneObject class. Use the Drawable property instead.
- Grid Colors property of the GraphChart class. Use the Grid Colors property instead.
- Grid Colors:X Color property of the GraphChart class. Use the Grid Colors: Major and Grid Colors: Minor properties instead.
- Grid Colors:Y Color property of the GraphChart class. Use the Grid Colors: Major and Grid Colors: Minor properties instead.
- Legend:Plots Shown property of the WaveformChart class. Use the Legend:Number of Rows property instead.
- Legend:Plots Shown property of the WaveformGraph class. Use the Legend:Number of Rows property instead.
- Pixel Width property of the ListBox class. Use the Bounds:Width property instead.
- Scrollbars Visible property of the Picture class. Use the Horizontal Scrollbar Visible and Vertical Scrollbar Visible properties instead.
- Set Geometry method of the SceneObject class. Use the Set Drawable method instead.
- Scene:Geometry:New Mesh method of the Application class. Use the Scene:Drawable:Geometry:New Mesh method instead.
- Drag Starting event of the Control class. Use the Drag Starting event of the appropriate control class instead.
- Drag Starting? event of the Control class. Use the Drag Starting? event of the appropriate control class instead.

Renamed Properties, Methods, and Events

The following properties, methods, and events are renamed in LabVIEW 8.5:

Class	LabVIEW 8.2 Name	LabVIEW 8.5 Name	Type
AbsTime, Numeric	Data Range	Data Entry Limits	Property
AbsTime, Numeric	Data Range:Increment	Data Entry Limits:Increment	Property

Class	LabVIEW 8.2 Name	LabVIEW 8.5 Name	Type
AbsTime, Numeric	Data Range:Maximum	Data Entry Limits:Maximum	Property
AbsTime, Numeric	Data Range:Minimum	Data Entry Limits:Minimum	Property
AbsTime, Numeric	Out of Range Action	Response to Value Outside Limits	Property
AbsTime, Numeric	Out of Range Action:Increment	Response to Value Outside Limits:Increment	Property
AbsTime, Numeric	Out of Range Action:Maximum	Response to Value Outside Limits:Maximum	Property
AbsTime, Numeric	Out of Range Action:Minimum	Response to Value Outside Limits:Minimum	Property
Application	Library:Get Project Library File Version	Library:Get File LabVIEW Version	Method
Application	Scene:Geometry:New Box	Scene:Drawable:Geometry: New Box	Method
Application	Scene:Geometry:New Cone	Scene:Drawable:Geometry: New Cone	Method
Application	Scene:Geometry:New Cylinder	Scene:Drawable:Geometry: New Cylinder	Method
Application	Scene:Geometry:New Height Field	Scene:Drawable:Geometry: New Height Field	Method
Application	Scene:Geometry:New Mesh	Scene:Drawable:Geometry: New Mesh	Method
Application	Scene:Geometry:New Sphere	Scene:Drawable:Geometry: New Sphere	Method
Application (ActiveX)	LibraryGetProjectLibFileV ersion	LibraryGetFileLVVersion	Method
Digital, NumericText, and Scale	Format & Precision	Display Format	Property
Digital, NumericText, and Scale	Format & Precision:Format	Display Format:Format	Property
Digital, NumericText, and Scale	Format & Precision:Precision	Display Format:Precision	Property

Class	LabVIEW 8.2 Name	LabVIEW 8.5 Name	Type
DigitalTable	Column Headers Visible	Signal Number Visible	Property
DigitalTable	Row Headers Visible	Transitions Visible	Property
SceneGraphDisplay and SceneWindow	Clear Color	Background Color	Property
SceneObject	Set Geometry	Set Drawable	Method
VI	Connector Pane	Connector Pane:Set	Property

LabVIEW MathScript Behavior Changes (Windows, Not in Base Package)

LabVIEW 8.5 includes the following changes to LabVIEW MathScript:

- Changes you make to the search path list or the working directory using the following MathScript functions apply only to the current instance of the **LabVIEW MathScript Window** or the MathScript Node from which you call the function:
  - `addpath`
  - `cd`
  - `path`
  - `rmpath`

  LabVIEW resets the search path list and the working directory to the default when you close the **LabVIEW MathScript Window** or when the VI that contains the MathScript Node stops running.
- The syntax for the `qz` function changed from `[q, z, alpha, beta, evec] = qz(a, b)` to `[S, T, Q, Z, R, L] = qz(A, B, type)`.

LabVIEW Class Icons

If you created a LabVIEW class icon in LabVIEW 8.2 and you want the icon displayed when you place a class control or indicator on the block diagram, you must update the class icon to occupy a smaller space so that the class mask does not obscure any part of the class icon. Use an image no larger than 32 pixels wide by 19 pixels high.

Opening LLBs in LabVIEW

The **Enable Windows Explorer for LLB files** option on the **Environment** page of the **Options** dialog box no longer exists. LabVIEW opens LLBs in the **LLB Manager** window. Refer to the National Instruments Web site at `ni.com/info` and enter the info code `exvfc5` for more information about opening LLBs.

Timed Loop Priority Level Restriction

In LabVIEW 8.2.*x* and earlier, you can select up to 2 to the power of 32 for the priority level of a Timed Loop. LabVIEW 8.5 supports only priority levels less than 65,535.

Waveform Data Type

When indexing beyond the bounds of an array of waveforms, the resulting waveform is a proper default waveform with the dt value equal to 1, instead of an improper waveform with the dt value equal to 0. This also is true when executing a For Loop with a scalar output tunnel zero times.

Upgrading from LabVIEW 8.0

You might encounter the following compatibility issues when you upgrade to LabVIEW 8.5 from LabVIEW 8.0. Refer to the *Upgrading from LabVIEW 8.2* 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 8.0 and 8.5 at `ni.com/manuals` for more information about the new features and changes in each version.

Platforms Supported

LabVIEW 8.2 and later includes the following changes in platforms supported:

- LabVIEW 8.2 and later does not support Windows XP x64.
- LabVIEW 8.2 and later does not support Mac OS X 10.3.8 or earlier.
- LabVIEW 8.2 provides some support for Macintosh computers with Intel processors. Refer to the National Instruments Web site at `ni.com/info` and enter the info code `macintel` for more information about Macintosh support. LabVIEW 8.5 provides support for Macintosh computers with both Intel and PowerPC processors.

System Requirements

(Windows) LabVIEW 8.2 and later requires at least 1.2 GB for the LabVIEW installation.

(Mac OS) LabVIEW 8.2 requires at least 500 MB of disk space for the minimum LabVIEW installation or 700 MB disk space for the complete LabVIEW installation. LabVIEW 8.5 requires at least 502 MB of disk space for the minimum LabVIEW installation or 734 MB disk space for the complete LabVIEW installation.

(Linux) LabVIEW 8.2 requires at least 430 MB of disk space for the minimum LabVIEW installation or 620 MB disk space for the complete LabVIEW installation. LabVIEW 8.5 requires at least 450 MB of disk space for the minimum LabVIEW installation or 640 MB disk space for the complete LabVIEW installation.

Printed Documentation

The following documents did not change for LabVIEW 8.2. Therefore, the content might not reflect changes made in LabVIEW 8.2.

- *LabVIEW Quick Reference Card*
- *LabVIEW Fundamentals Manual*—Because the *LabVIEW Fundamentals Manual* is a subset of the **Fundamentals** book in the *LabVIEW Help*, refer to the **Fundamentals** book on the **Contents** tab of the *LabVIEW Help* for updated content.

VI and Function Behavior Changes

The behavior of the following VIs and functions changed in LabVIEW 8.2 and later.

Communicating between Application Instances

In LabVIEW 8.2 and later, you cannot use the Obtain Queue, Obtain Notifier, Create User Event, Create Semaphore, and Create Rendezvous functions to communicate between LabVIEW application instances. If you obtain or create a queue, notifier, user event, semaphore, or rendezvous reference in one application instance, you cannot use that reference in another application instance.

Back Transform Eigenvectors VI

The **index low**, **index high**, and **Scale** inputs of the Back Transform Eigenvectors VI are required inputs.

DataSocket Write Function

In LabVIEW 8.0.1, the default behavior for DataSocket Write function changed to asynchronous. If you have LabVIEW 8.0 and LabVIEW 8.2 or later installed on your computer, the DataSocket API Client VI example in the `labview\examples\Shared Variable` directory returns an error when you stop the VI. You must update LabVIEW 8.0 to LabVIEW 8.0.1 to use this example in LabVIEW 8.2 or later.

File I/O VIs

The Write To Spreadsheet File VI and Read From Spreadsheet File VI are polymorphic VIs. The Write to Spreadsheet File VI adapts to the value you wire to the **format** input. The Read From Spreadsheet File VI includes the following instances: DBL, I64, and string.

GPIB Status Function

In LabVIEW 8.0, the GPIB Status function did not execute if the **error in** input received an error. In LabVIEW 8.2 or later, the GPIB Status function always executes, even if the **error in** input receives an error.

Histogram VI

The default for the **intervals** input of the Histogram VI changed to `10`.

Open VI Reference Function

The default behavior for the **options** input of the Open VI Reference function is to prompt users to find missing subVIs of the referenced VI. A new value, `0x20`, specifies not to display the **Find** dialog box or prompt users to find missing subVIs of the referenced VI.

Polynomial Roots VI

If **P(x)** equals a nonzero constant, the Polynomial Roots VI does not return an error. However, if **P(x)** equals `0`, the Polynomial Roots VI returns error `-20111`. The input polynomial coefficients for this VI cannot all be zeros.

Ramp Pattern VI

In the Ramp Pattern VI, if **samples** is `1` and **exclude end?** is TRUE, the VI returns an array with one element of **start**, with no error. In LabVIEW 8.0, the VI returned an error with these conditions.

Read Registry Value Simple VI

LabVIEW 8.0 incorrectly handled `REG_MULTI_SZ` string formatting, which the VI used for a flattened array of strings. This issue required you to write a parser to handle this type of data for the Read Registry Value Simple VI. In LabVIEW 8.2 and later, the Read Registry Value Simple VI returns this type of data in the same format used in the Write Registry Value Simple VI. You no longer need to add your own parser. Using your own parser with these VIs in LabVIEW 8.2 and later causes the Read Registry Value Simple VI to return bad data.

Resample Waveforms (single-shot) VI

The default value of the **open interval?** input of the Resample Waveforms (single shot) VI changed from TRUE to FALSE, which selects a closed interval. If you do not update existing code accordingly, the VI might not return the expected result.

Sound VIs

In the Sound Input Read and the Sound File Read Simple VIs, the t0 component of the data output returns the time stamp for the first sample read. LabVIEW approximates the initial time that it reads the first sample.

Calling The Sound Output Stop VI no longer is necessary to stop the sound on a continuous sound task.

The Sound Output Wait VI works in **Continuous Samples** mode and in **Finite Samples** mode.

Waveform VIs

LabVIEW 8.2 and later includes changes to the following Waveform VIs:

- **Basic Level Trigger Detection**—In both instances of this VI, the **slope** input changed to **trigger slope**.
- **Get Waveform Subset**—Includes the following instances: WDT Get Waveform Subset DBL, WDT Get Waveform Subset CDB, WDT Get Waveform Subset EXT, WDT Get Waveform Subset I16, WDT Get Waveform Subset I32, WDT Get Waveform Subset I8, and WDT Get Waveform Subset SGL. The start/duration format input no longer includes an **Absolute Time** option. The start input changed to start samples/time, and the actual start output changed to actual start samples/time.
- **Get Waveform Time Array**—The X array output changed from a double-precision, floating-point numeric data type to a time stamp data type.
- **Get Y Value**—This VI and the corresponding polymorphic instances were renamed to Get XY Value. The Get XY Value VI now includes an X value output, and the **data value** output changed to Y value.
- **Number of Waveform Samples**—This VI is a polymorphic VI with the following instances: WDT Number of Waveform Samples DBL, WDT Number of Waveform Samples CDB, WDT Number of Waveform Samples EXT, WDT Number of Waveform Samples I16, WDT Number of Waveform Samples I32, WDT Number of Waveform Samples I8, and WDT Number of Waveform Samples SGL.
- **Read Waveform from File**—Returns an error status of TRUE in the **error out** output when the error is end-of-file.
- **Replace Subset**—The start input changed to start samples/time, and the actual start value output changed to actual start samples/time.
- **Search for Digital Pattern**—The start input changed to start index/time.

- **Search Waveform**—The time of best fit and time of fits outputs changed from a double-precision, floating-point numeric data type to a time stamp data type.
- **Waveform Min Max**—The min time and max time outputs changed from a double-precision, floating-point numeric data type to a time stamp data type.
- **Waveform to XY Pairs**—The **x** element of the XY pairs output changed from a double-precision, floating-point numeric data type to a time stamp data type.

Property, Method, and Event Behavior Changes

The behavior of the following properties, methods, and events changed in LabVIEW 8.2 and later:

- The default behavior for the **options** input of the ActiveX GetVIReference method is to prompt users to find missing subVIs of the referenced VI. A new value, `0x20`, specifies not to display the **Find** dialog box or prompt users to find missing subVIs of the referenced VI.
- The Add Item method of the ProjectItem class return an error when you try to add a shared variable to a library that is not opened in a project.
- If the **Auto Dispose Ref** input of the Run VI method is TRUE and the method returns an error, LabVIEW does not dispose of the reference.
- Valid values for the Application:Language property include `zh-cn` to indicate that Simplified Chinese is the language of the LabVIEW environment.
- In LabVIEW 8.0, .NET methods that pass array data types by reference pass all data as the refnum data type. In LabVIEW 8.2 and later, .NET methods that pass array data types by reference pass the data as the actual data type.
- The Edit Position property of the DigitalTable, MulticolumnListbox, Table, and TreeControl classes returns values of (–2, –2) to indicate that the user is not making edits to the text of the control. The Edit Row property of the ListBox class returns a value of –2 to indicate that the user is not making edits to the text of the control.
- In LabVIEW 8.0, the Defer Panel Updates property did not defer the update of front panels in a subpanel. In LabVIEW 8.2 and later, the Defer Panel Updates property works with subpanels.
- The Application Instance Close and Application Instance Close? events replace the Application Exit and Application Exit? events. When you use the Application Instance Close event in a VI running outside a LabVIEW project, LabVIEW generates the event when you

quit LabVIEW through the user interface or programmatically. LabVIEW generates the Application Instance Close? event when you quit LabVIEW through the user interface. When you register the Application Instance Close and Application Instance Close? events for a VI running within a LabVIEW project, LabVIEW generates the events when the application instance closes or when you quit LabVIEW.

Deprecated Properties, Methods, and Events

LabVIEW 8.2 and later does not support the following properties, methods, and events.

- LabVIEW 8.2 and later does not support the Connector Pane property.
- LabVIEW 8.*x* does not support the Data Type property in the Variable class. Use the Data Type (Variant) property in the Variable class instead.

Renamed Properties, Methods, and Events

The following properties, methods, and events are renamed in LabVIEW 8.2 and later:

Class	**LabVIEW 8.0 Name**	**LabVIEW 8.2 and Later Name**	**Type**
Application	Disconnect From Slave	LVRT:Disconnect From Slave	Method
Application	Application Exit	Application Instance Close	Event
Application	Application Exit?	Application Instance Close?	Event
IntensityGraph, MixedSignalGraph, and WaveformGraph	Cursor Palette Visible	Cursor Legend Visible	Property
Library	Delete Library Tag	Library Tag:Delete	Method
Library	Get Icon	Icon:Get	Method
Library	Get Library Tag	Library Tag:Get	Method
Library	Get Library Tag Names	Library Tag:Get Names	Method
Library	Get Lock State	Lock State:Get	Method
Library	Get Source Scope	Source Scope:Get	Method

Class	LabVIEW 8.0 Name	LabVIEW 8.2 and Later Name	Type
Library	Save	Save:Library	Method
Library	Save a Copy	Save:Copy	Method
Library	Set Icon	Icon:Set	Method
Library	Set Library Tag	Library Tag:Set	Method
Library	Set Lock State	Lock State:Set	Method
Library	Set Source Scope	Source Scope:Set	Method
Listbox, MulticolumnListbox, and TreeControl	Drag/Drop:Allow Item Dragging	Drag/Drop:Allow Dragging	Property
Path and String	Allow Drop	Allow Dropping	Property
ProjectItems	Delete Tag	Tag:Delete	Property
ProjectItems	Get Tag	Tag:Get Tag	Property
ProjectItems	Get Tag Names	Tag:Get Names	Property
ProjectItems	Get XML Tag	Tag:Get XML Tag	Property
ProjectItems	Set Tag	Tag:Set Tag	Property
ProjectItems	Set XML Tag	Tag:Set XML Tag	Property
ProjectItems	Library Item Type String	Library Item Type:String	Property
ProjectItems	Library Item Type	Library Item:Type	Property

Application Builder Changes

In LabVIEW 8.2 and later, you cannot view the contents of a stand-alone application (EXE) or shared library (DLL) by renaming the application or shared library to have a `.llb` file extension. You also cannot access a VI in a stand-alone application or shared library by specifying the path to the VI from outside of the application or shared library. Refer to the National Instruments Web site at `ni.com/info` and enter the info code `exjk3b` for more information about viewing and accessing applications and shared libraries.

Upgrading from LabVIEW 7.*x*

You might encounter the following compatibility issues when you upgrade to LabVIEW 8.5 from LabVIEW 7.*x*. Refer to the *Upgrading from*

LabVIEW 8.0 and *Upgrading from LabVIEW 8.2* 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 7.*x* and 8.5 at `ni.com/manuals` for more information about the new features and changes in each version.

Note The *LabVIEW Quick Reference Card* and the *LabVIEW Fundamentals Manual* did not change for LabVIEW 8.2 or 8.5. You can access the PDF versions of these documents in the `labview\manuals` directory. Refer to the *Upgrading from LabVIEW 8.0* section of this document for more information about these documents.

Platforms Supported

LabVIEW 8.*x* includes the following changes in platforms supported:

- LabVIEW 7.1 and later do not support Windows Me/98/95. LabVIEW 8.*x* does not support Windows NT.
- LabVIEW 8.*x* does not support Mac OS X 10.2 or earlier.
- LabVIEW 8.*x* 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 8.5 requires a minimum of 256 MB of RAM, but National Instruments recommends 1 GB 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.*x* 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.*x* 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.

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.*x* 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 8.5 requires at least 502 MB of disk space for the minimum LabVIEW installation or 734 MB disk space for the complete LabVIEW installation.

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.*x* 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.

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 8.5 requires at least 450 MB of disk space for the minimum LabVIEW installation or 640 MB disk space for the complete LabVIEW installation.

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.*x* requires GNU C Library version 2.2.4 or later.

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.*x* 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.*x* does not support custom palette views. You can edit a palette set without using a custom palette view. Refer to the National Instruments Web site at `ni.com/info` and enter the info code `lv8palette` 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 7.1 or 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.*x*. 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.*x* 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.*x*, 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.*x*, 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.*x* 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 yi* is the complex conjugate of yi.

Easy Text Report VI (Mac OS and Linux)

The connector pane of the Easy Text Report VI changed. In LabVIEW 8.*x*, 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`, `%b`, 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.*x*, 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.*x*, 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.*x*, LabVIEW coerces 32-bit integer inputs to 16-bit integers.

Mathematics VIs and Matrices

In LabVIEW 8.*x*, **Mathematics** VIs support the matrix data type. If you load a VI from LabVIEW 7.*x* in LabVIEW 8.*x* and the VI contains a Mathematics VI wired to a function that can use the matrix data type and is instead using a 2D array, a red 7.*x* glyph appears on the function. The red glyph indicates that LabVIEW replaced the 2D array with the matrix data type.

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.*x*, 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.*x*, 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.*x*, 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 VI name 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. In LabVIEW 8.5, an error occurs if LabVIEW cannot find the file or if the VI name of the file is the same as the qualified VI name as another VI in memory on that target.

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.*x*, 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 string instance of 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.*x*, the string instance of 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.*x*, 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 run when the error in input receives an error. In LabVIEW 8.*x*, these VIs attempt to run even if the error in input receives an error. However, if you open a LabVIEW 7.*x* VI in LabVIEW 8.*x*, LabVIEW updates the VI in order to maintain 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.*x*, 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.*x*, 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.*x*, 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.*x* because LabVIEW 8.0 and later 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 *LabVIEW Help* for more information about application instances.

Deprecated VIs and Functions

LabVIEW 8.*x* 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.
- **(Mac OS)** LabVIEW 7.1 and later do not install the PPC VIs. Use the TCP VIs instead.
- LabVIEW 8.*x* does not support the QR Factorization VI. Use the QR Decomposition VI instead.
- LabVIEW 8.*x* does not support the Levenberg Marquardt or the Nonlinear Lev-Mar Fit VIs. Use the Nonlinear Curve Fit VI instead.
- In LabVIEW 8.*x*, the VISA Status Description function is not on the **Functions** palette. Use the Simple Error Handler or General Error Handler VIs instead.
- LabVIEW 8.*x* 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.*x* 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.*x*, 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.*x*, 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.*x*, 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.*x*, 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.*x*, 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.*x*, the Pulse Parameters VI is not on the **Functions** palette. Use the Transition Measurements VI instead to measure the **slew rate**, **duration**, **overshoot** (the Transition Measurements VI

equivalent is the **post-transition** output), or **preshoot** (the Transition Measurements VI equivalent is the **pre-transition** output) 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.*x*, 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.*x*, the NI DIAdem Report Wizard Express VI is not on the **Functions** palette. Use the DIAdem Report Express VI instead.
- In LabVIEW 8.*x*, 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.*x*, 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 and Linux)** In LabVIEW 8.*x*, 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.*x* instead. The examples shipped with LabVIEW 7.*x* do not ship with LabVIEW 8.*x*.

File I/O VIs and Functions

In LabVIEW 8.*x*, the Read Characters From File VI is not on the **Functions** palette. Use the Read from Text File function instead.

In LabVIEW 8.*x*, 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 file dialog box.
- 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.*x*, 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.*x*, the Write Characters To File VI is not on the **Functions** palette. Use the Write to Text File function instead.

In LabVIEW 8.*x*, the Access Rights function is not on the **Functions** palette. Use the Get Permissions and Set Permissions functions instead.

In LabVIEW 8.*x*, the EOF function is not on the **Functions** palette. Use the Get File Size and Set File Size functions instead.

In LabVIEW 8.*x*, the List Directory function is not on the **Functions** palette. Use the List Folder function instead.

In LabVIEW 8.*x*, 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.*x* 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.*x*, the Seek function is not on the **Functions** palette. Use the Get File Position and Set File Position functions instead.

In LabVIEW 8.*x*, 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.*x*, the Volume Info function is not on the **Functions** palette. Use the Get Volume Info function instead.

In LabVIEW 8.*x*, 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.*x*, 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.

Property, Method, and Event Behavior Changes

The behavior of the following properties, methods, and events changed in LabVIEW 7.1 or 8.0.

Application Properties and Methods

In LabVIEW 8.*x*, 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 *LabVIEW Help* 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.*x*. If you have VIs that use the Old Open FP method from LabVIEW 7.0, replace the method with the Front Panel:Open method.

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. If the Run VI Method generates an error, the reference is not automatically closed. In LabVIEW 8.0, LabVIEW automatically 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. In LabVIEW 8.2 and later, the behavior has been changed back to the 7.1 behavior.

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.*x*, 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.*x* 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.*x*, the number of visible items in the listbox does not limit the row number you can wire to this property.

LabVIEW 8.*x* 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.*x*, 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.*x*, 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 one less pixel for each line in the tree control than in LabVIEW 7.*x*. For example, if you load a LabVIEW 7.x VI in LabVIEW 8.x that contains a property node which returns the height and width of a tree control as 70 pixels and 16 pixels, any new property nodes you place to determine height and width return 69 pixels and 15 pixels.

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.*x* when you use the VI Strings:Import method.

Deprecated Properties, Methods, and Events

LabVIEW 8.*x* does not support the following properties, methods, and events.

Cursor Properties

LabVIEW 8.*x* does not support the Cursor Lock Style property. Use the Cursor Mode property instead.

ListBox, MulitcolumnListbox, Table, DigitalTable, and TreeControl Properties and Events

LabVIEW 8.*x* does not support the Cell Foreground Color property for multicolumn listboxes. Use the Active Cell:Cell Font:Color property instead.

LabVIEW 8.*x* 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.*x* does not support the Active Cell Properties:Foreground Color property for tree controls. Use the Active Cell:Cell Font:Color property instead.

LabVIEW 8.*x* 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.*x* 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.*x* does not support the Color property in the Panel class. If you use this property in LabVIEW 8.*x*, the property applies only to the upper-leftmost pane. Use the Pane Color property in the Pane class instead.

Subpanel Properties

In LabVIEW 8.*x*, 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.*x* does not support the X Scrollbar Visible property for subpanel controls. Use the Horizontal Scrollbar Visibility property for panes instead.

LabVIEW 8.*x* does not support the Y Scrollbar Visible property for subpanel controls. Use the Vertical Scrollbar Visibility property for panes instead.

LabVIEW 8.*x* 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.*x* does not support the Front Panel Window:Auto Center property. Use the Front Panel:Center method instead.

LabVIEW 8.*x* does not support the Front Panel Window:Size to Screen property. Use the Front Panel Window:State property instead.

LabVIEW 8.*x* does not support the Front Panel Window:Origin property in the VI class. If you use this property in LabVIEW 8.*x*, the property applies only to the upper-leftmost pane. Use the Origin property in the Pane class instead.

LabVIEW 8.*x* does not support the Front Panel Window:Show Scroll Bars property in the VI class. If you use this property in LabVIEW 8.*x*, 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.*x* 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.*x*, the methods apply only to the upper-leftmost pane. Use the Get Scaling Mode and Set Scaling Mode methods in the Pane class instead.

In LabVIEW 8.*x* you cannot select the Mouse Down, Mouse Down?, Mouse Move, or Mouse Up events of the VI class in the **Edit Events** dialog box. Use the Mouse Down, Mouse Down?, Mouse Move, and Mouse Up events in the Pane class, respectively, instead.

Application Item Tags

The following list includes application item tags that have been removed from LabVIEW either because the feature is no longer available or because it has been combined with another feature:

- `APP_SAVE_WITH_OPTIONS`
- `APP_UPDATE_VXI`

- `APP_DSC_TOOLBAR`
- `APP_DSC_TAGEDITOR`
- `APP_DSC_TAGMONITOR`
- `APP_DSC_HTV`
- `APP_DSC_ENGINE`
- `APP_DSC_SECURITY`
- `APP_DSC_LOGOUT`
- `APP_DSC_CPWD`
- `APP_DSC_USERINFO`
- `APP_DSC_USEREDITOR`
- `APP_DSC_ADVANCED`
- `APP_DSC_STARTUP`
- `APP_DSC_SRVBRW`
- `APP_DSC_IST`
- `APP_DSC_IMAGENAV`
- `APP_DSC_OPTIONS`
- `APP_SRC_CODE_CTRL`
- `APP_BUILD_STANDALONE_APP`
- `APP_EDIT_VI_LIBRARY`
- `APP_DN_ASSEMBLY_REFS`
- `APP_SHOW_CLIPBOARD`
- `APP_VIEW_PRINTED_MANUALS`
- `APP_RT_ENGINE_INFO`
- `APP_SWITCH_EXEC_TARGET`
- `APP_REALTIME`

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.*x* 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.*x*. If an application uses HiQ VIs, consider replacing them with 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.*x*, 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 *LabVIEW Help* for more information about application instances.

VI String File Syntax

LabVIEW 8.*x* 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.*x*, you cannot import VI string files saved in previous versions of LabVIEW.

LabVIEW 7.1 and earlier lists listbox strings in the `<ITEMS>` section of its private data. LabVIEW 8.*x* 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.*x*, the listbox can have multiple fonts, which LabVIEW lists in the `<CELL_FONTS>` section of its private data.

LabVIEW 7.1 and earlier lists multicolumn listbox strings in its default data. However, the default data for a multicolumn listbox is an integer or array of integers. LabVIEW 8.*x* lists multicolumn listbox strings in its private data.

LabVIEW 7.1 and earlier exports neither strings nor fonts for tree controls. LabVIEW 8.*x* 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.*x*, each line of an export file contains no more than two tags for private or default data. LabVIEW 8.*x* also indents items once for each nesting level.

Complete the following steps to convert VI string files to the LabVIEW 8.*x* format.

1. Import the VI string file in the previous version of LabVIEW.
2. Save the VI.
3. Load the VI in LabVIEW 8.*x*.
4. Select **Tools»Advanced»Export Strings** to save the VI string file in the LabVIEW 8.*x* format.

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

The format in which LabVIEW stores type descriptors changed in LabVIEW 8.*x*. LabVIEW 7.*x* stores type descriptors in 16-bit flat representation. LabVIEW 8.*x* stores type descriptors in 32-bit flat representation. This change eliminates the 64 KB size limitation of type descriptors.

LabVIEW 8.*x* 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.*x* 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.*x*, when you load a VI last saved in LabVIEW 7.*x* or earlier, LabVIEW 8.*x* 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.*x* 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.*x* 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.

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.*x*. 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/info` and enter the info code `exgucn` 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.

   Refer to the **Fundamentals»Organizing and Managing a Project» How-to»Using Source Control in LabVIEW** book on the **Contents** tab of the *LabVIEW Help* for information about configuring LabVIEW to work with a 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.

   Refer to the **Fundamentals»Organizing and Managing a Project» How-to»Creating a LabVIEW Project** book on the **Contents** tab of the *LabVIEW Help* for information about creating a LabVIEW project.

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.*x*, 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.*x* removes any code that does not execute. Therefore, if you load a VI in LabVIEW 8.*x* 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.*x*, 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.*x*, 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. Right-click the DLL in the **Project Explorer** window, select **Properties** from the shortcut menu, select **Advanced** from the **Category** list, 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.*x*, 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.*x*, 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.*x*.

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.*x*. Refer to the *LabVIEW Help* for more information about linking shared libraries to `labviewv.lib`.

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.*x*, 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 8.5 from LabVIEW 6.*x*. Refer to the *Upgrading from LabVIEW 7.x*, *Upgrading from LabVIEW 8.0*, and *Upgrading from LabVIEW 8.2* 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 6.*x* and 8.5 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 serial 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 menu, 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 7.*x* and later.

Unit	6.1 Definition	7.*x*and Later 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 7.0 and later, 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 7.*x* and later, 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.*x*, 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.

Displaying the Front Panel When Loaded

In LabVIEW 7.*x* and later, 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 instantiates 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 7.1, 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. In LabVIEW 8.0 and later, if you use the Open VI Reference function to create a reference to a template, LabVIEW instantiates a VI from the template even if that template is already in memory.

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* or Earlier Versions

Refer to the National Instruments Web site at `ni.com/info` and enter the info code `ext8h9` for information about upgrading to LabVIEW 8.5 from LabVIEW 5.*x* or earlier.

LabVIEW 8.5 Features and Changes

Refer to the *LabVIEW Help* for more information about LabVIEW 8.5 features, including programming concepts, step-by-step instructions, and reference information. Access the *LabVIEW Help* by selecting **Help» Search the LabVIEW Help**.

Refer to the `readme.html` in the `labview` directory for known issues, a partial list of bugs fixed, additional compatibility issues, and information about late addition features in LabVIEW 8.5.

Installing LabVIEW

LabVIEW 8.5 offers the following installation options on Windows.

- **DVD**—Includes LabVIEW, the device drivers, and LabVIEW SignalExpress.
- **CD**—Includes LabVIEW only. If you install LabVIEW using the CD, you install the device drivers and LabVIEW SignalExpress using the National Instruments Device Drivers CD.

The Measurement & Automation Explorer (MAX) no longer appears in the tree during installation if you install using the installation CD. The components of MAX that LabVIEW requires are installed automatically. You can install MAX from the National Instruments Device Drivers CD.

LabVIEW Documentation

LabVIEW 8.5 includes the following documentation enhancements:

- The `readme.html` includes a partial list of user-reported bugs fixed in the current version of LabVIEW. Refer to the *Bug Fixes* section of the `readme.html` in the `labview` directory to review the list of bugs.
- Reference topics in the *LabVIEW Help* include LabVIEW module-specific information if the module is installed and if any module-specific information exists for the VI, function, and so on exists.

New Example VIs

Refer to the **New Examples for LabVIEW 8.*x*** folder on the **Browse** tab of the NI Example Finder to view descriptions for and launch example VIs added to LabVIEW 8.*x*.

Block Diagram Enhancements

LabVIEW 8.5 includes the following enhancements to the block diagram and related functionality.

Creating Shared Reentrant VIs

When you configure a reentrant VI in LabVIEW 8.2.*x* and earlier, LabVIEW preallocates a clone of the reentrant VI in memory for each call to the reentrant VI. In LabVIEW 8.5, you can configure a reentrant VI to share the clone VI among callers without preallocating a clone VI in memory for each call to the reentrant VI. Sharing clone VIs reduces memory usage because LabVIEW does not preallocate a clone for each call to the reentrant VI.

To configure a VI to share the clone VI among callers, select **File»VI Properties** and select **Execution** from the **Category** pull-down menu to display the **Execution Properties** page of the **VI Properties** dialog box. Place a checkmark in the **Reentrant execution** checkbox and select the **Share clones between instances** option to share the clone VIs.

However, sharing clone VIs can reduce VI execution speed. Do not select the **Share clones between instances** option if you want to preserve state information across calls to a clone VI, such as uninitialized shift registers. To preserve state information across calls to a clone VI, select the **Preallocate clone for each instance** option on the **Execution Properties** page.

Conditional and Diagram Disable Structure Enhancements

When LabVIEW loads a VI with user-defined objects, such as subVIs and type definitions, in the disabled subdiagram of a Diagram Disable structure or in the inactive subdiagrams of a Conditional Disable structure, LabVIEW does not load these objects into memory. However, when you display the block diagram of the VI, if LabVIEW cannot find the objects, the missing objects appear with a question mark icon. The VI does not break because LabVIEW does not include the code when compiling and executing the VI.

Front Panel Enhancements

LabVIEW 8.5 includes the following enhancements to the front panel and related functionality.

Digital Waveform Graph Enhancements

The default view for the plot legend of the digital waveform graph is the tree view, located to the left of the plot area of the graph. The tree view displays both digital lines and buses by default. You can expand and contract digital buses in the tree view of the plot legend by clicking the expand/contract symbol to the left of the digital bus. You can use the standard view of the plot legend to achieve the appearance of the plot legend in LabVIEW 8.2 and earlier.

By default, the digital waveform graph displays both digital lines and buses in the plot area. Customize the plot area of the digital waveform graph to display only the data you want to view.

Export Simplified Image Enhancements

The **Export Simplified Image** on the shortcut menu of graphs, charts, tables, picture controls, digital data, and digital waveform controls moved from the **Data Operations** shortcut menu to the top-level shortcut menu.

You also can right-click a graph, chart, table, picture control, digital data, or digital waveform control or indicator contained within an XControl and select **Export Simplified Image** from the shortcut menu to export an image to the clipboard or to save an image to disk.

Front panel images exported using the **Export Simplified Image** menu item no longer export with a border enclosing the image.

Plot Legend Enhancements

Customize how a plot appears in the plot area of a graph or chart by clicking the glyph that corresponds to that plot in the plot legend and select from the shortcut menu options.

You can add a vertical or horizontal scroll bar to the plot legend of graphs and charts. Use a scroll bar to view plots in a plot legend without exposing all the plots at any one time.

To add a scroll bar to a plot legend, right-click the plot legend and select **Visible Items»Horizontal Scrollbar** or **Visible Items»Vertical Scrollbar** from the shortcut menu.

Refer to the *Customizing Graphs and Charts* topic in the **Fundamentals» Graphs and Charts»Concepts** book on the **Contents** tab in the *LabVIEW Help* for more information about plot legends and plot legend scroll bars.

Using a Mixed Checkbox

Use a mixed checkbox, located on the **System** palette, to display a TRUE, FALSE, or MIXED value. For example, use a mixed checkbox if you want to display a set of Boolean values in a single indicator, where that set of Boolean values are either all TRUE, all FALSE, or a combination of TRUE and FALSE, called MIXED. Use the mixed checkbox control to simultaneously change a set of Boolean values to either a TRUE or FALSE value.

Although the mixed checkbox is an enumerated type control, you can configure the control similarly to a Boolean control.

To disable the ability to set the MIXED value of a mixed checkbox interactively, right-click the control and select **Allow Mixed** from the shortcut menu to remove the checkmark next to the menu item. When you disable **Allow Mixed**, LabVIEW can assign the value of the mixed checkbox as MIXED, but you cannot operate to MIXED at run time by clicking on it.

Refer to the *Customizing Graphs and Charts* topic in the **Fundamentals» Graphs and Charts** book on the **Contents** tab in the *LabVIEW Help* for more information about the digital waveform graph.

Miscellaneous Front Panel Enhancements

You can configure a front panel control to scale automatically while the pane resizes. Right-click a splitter bar and select **Pane Sizing»Scale Objects While Resizing** from the shortcut menu.

Environment Enhancements

LabVIEW 8.5 introduces the following enhancements to the LabVIEW environment.

Automatic Saving for Recovery Enhancements

In the event of an irregular shutdown or system failure, LabVIEW 8.5 now backs up modified project `(.lvproj)`, project library `(.lvlib)`, XControl `(.xctl)`, and LabVIEW class `(.lvclass)` files open at the time of the shutdown or failure to a temporary location. These file types appear with any recovered VI `(.vi)`, VI template `(.vit)`, control `(.ctl)`, and control template`(.ctt)` files in the **Select Files to Recover** window the next time you launch LabVIEW. To enable the **Compare Backup with Original** button in the **Select Files to Recover** window for project, project library, XControl, and LabVIEW class files, you manually must configure text comparison for file recovery.

Saving for Multiple Previous Versions of LabVIEW

You can save VIs, projects, and project libraries for a previous version of LabVIEW to make upgrading LabVIEW convenient and to help you maintain files in more than one version of LabVIEW when necessary. Select **File»Save for Previous Version** to open the **Save for Previous Version** dialog box. Choose **8.5**, **8.2**, or **8.0** from the **LabVIEW Version** pull-down menu to save for a previous version of LabVIEW.

Dialog Box Enhancements

LabVIEW 8.5 includes the following dialog box enhancements.

Add Targets and Devices Dialog Box Enhancements

If you add a target to a LabVIEW project that conflicts with another existing target, the **Add Targets and Devices** dialog box displays the **Options** column next to the **Targets and Devices** list. Click the down arrow next to the conflict resolution in the **Options** column and select from the available options to resolve the conflict. You cannot add a target until you resolve all conflicts in the **Options** column for the selected target.

Call Library Function Dialog Box Enhancements

Right-click a Call Library Function Node on the block diagram and select **Configure** from the shortcut menu to display the **Call Library Function** dialog box. The **Error Checking** page provides several options for integrated error checking with the Call Library Function Node.

- The **Maximum** and **Default** controls on the **Error Checking** page allow LabVIEW to recover from unhandled exceptions that occur in the configuration of the Call Library Function Node or during a call to a shared library or DLL.

- The **Disabled** control on the **Error Checking** page disables error checking but improves the execution speed of the Call Library Function Node. However, certain errors can cause an irregular shutdown of LabVIEW. Before disabling error checking, be sure that the function the Call Library Function Node references does not raise any unhandled exceptions.

The **Call Library Function** dialog box also includes the following miscellaneous enhancements:

- On the **Parameters** page, the **Minimum size** control displays **<None>** by default. The **Minimum size** control is visible when you select **String** from the **Type** pull-down menu and then select **C String Pointer** from the **String format** pull-down menu or when you select **Array** from the **Type** pull-down menu and then select **Array Data Pointer** from the **Array format** pull-down menu.
- On the **Function** page, the **Reentrant** control is renamed **Run in any thread**.

Options Dialog Box Enhancements and Changes

LabVIEW 8.5 includes the following **Options** dialog box enhancements:

- On the **Block Diagram** page, the **Use transparent name labels** is enabled by default.
- On the **Environment** page, the **Enable Windows Explorer for LLB files** option no longer exists.
- On the **Front Panel** page, the components are in order based on items that apply to edit time behavior and items that apply to run-time behavior.
- On the **Front Panel** page, put a checkmark in the **Connector pane terminals default to Required** checkbox to set new connector pane terminals to required.
- The **New and Changed for LabVIEW 8.x** page contains the following link which opens the *LabVIEW Help*: **View the complete list of New and Changed for LabVIEW 8.x**. The *Complete List of New and Changed for LabVIEW 8.x Options* topic includes new and changed features from LabVIEW 7.1 to 8.5. The **New and Changed for LabVIEW 8.x** page now includes only the components added or changed since the previous release of LabVIEW.
- On the **Printing** page, the default for MacIntel platform is standard printing. With the exception of **Margins**, all other printing options for the MacIntel no longer exist.

Properties Dialog Box Enhancements and Changes

LabVIEW 8.5 includes a new **Numeric Node Properties** dialog box. Use this dialog box to configure output settings for **Numeric** functions, such as the data type of the output value. If the data type of the output value is fixed-point, you also can configure how the function handles overflow and quantization conditions.

LabVIEW 8.5 includes the following enhancements to the pages of the **Properties** dialog boxes for controls, indicators, and constants:

- The **Display Format** page replaces the **Format & Precision** page.
- The **Data Entry** and **Data Type** pages replace the **Data Range** page.
- The **Representation** section on the **Data Type** page includes the **FXP** (**Fixed-point**) option.
- The **Data Type** page includes **Fixed-Point Configuration** options.
- The **Use Default Limits** checkbox replaces the **Use Default Range** checkbox on the **Data Entry** page.
- The **Size** section on the **Appearance** page includes the height and width, in pixels, of controls and indicators.

Miscellaneous Dialog Box Enhancements

LabVIEW 8.5 includes the following miscellaneous dialog box changes:

- On the second page of the **Import Web Service** wizard, you can input the web service authentication or proxy server information.
- The **Input the project library and destination directory** page replaces the **Input project library information and destination directory** page of the **Import Web Service** wizard.
- On the **General Settings** page of the **Project Library Properties**, **Class Properties**, and **XControl Properties** dialog boxes, you can click the **Enter Password** button to unlock a password-protected library.
- The **Authentication** dialog box includes two options to skip prompts for passwords you do not know when you attempt to access multiple password-protected VIs in a single operation. Click the **Skip** button to skip a password prompt for a single VI, or place a checkmark in the **Skip remaining password prompts for this operation** checkbox and click the **Skip** button to skip all password-protected VIs without cancelling the operation.
- On the **Window Size** page of the **VI Properties** dialog box, the **Set to Current Window Size** button changed to **Set to Current Panel Size**.
- On the **Plots** page of the **Waveform Graph Properties** dialog box, the **Do not use waveform names for plot names** checkbox is now the **Ignore waveform or dynamic attributes, including plot names**

checkbox. This checkbox is available only for graphs and charts with dynamic or waveform data.

Palette Enhancements

LabVIEW 8.5 includes the following palette enhancements:

- You can make palette categories visible or hidden in all palette view formats: **Category (Standard)**, **Category (Icons and Text)**, **Icons**, **Icons and Text**, **Text**, and **Tree**. On a pinned palette, click the **View** button on the palette toolbar and select **Change Visible Categories** from the shortcut menu to display all categories on the **Controls** or **Functions** palette.
- The **Application Control VIs and Functions** palette includes the **Memory Control** palette.

Miscellaneous Environment Enhancements

Any time you print in LabVIEW, the printer you use becomes the default printer in LabVIEW. LabVIEW does not recognize changes to the system default printer until you restart LabVIEW.

LabVIEW Project Enhancements

LabVIEW 8.5 includes the following enhancements to the LabVIEW project and related functionality.

Shared Variable and NI-PSP Enhancements

LabVIEW 8.5 includes the following shared variable and NI-PSP enhancements.

Absolute and Target-Relative Shared Variable Nodes

You can set a Shared Variable node as absolute or target-relative depending on how you want the node to connect to the variable. When you create a shared variable from a target in a LabVIEW project, any Shared Variable nodes you create from the shared variable are absolute by default.

Note You can create, configure, and host shared variables on Windows systems only. You can use the **DataSocket** VIs and functions to read or write shared variables from other platforms. If you have a VI with a Shared Variable node that was configured on a Windows system, you also can move that VI to another platform.

An absolute Shared Variable node connects to the shared variable on the target on which you created the variable. A target-relative Shared Variable node connects to the shared variable on the target on which you run the VI that contains the node. If you move a VI that contains a target-relative

Shared Variable node to a new target, you also must move the shared variable to the new target. Use target-relative Shared Variable nodes when you expect to move VIs and variables to other targets.

To change an absolute Shared Variable node to target-relative, right-click the Shared Variable node on the block diagram and select **Change to Target Relative** from the shortcut menu. To change a target-relative Shared Variable node to absolute, right-click the Shared Variable node and select **Change to Absolute** from the shortcut menu.

TCP Implementation of NI-PSP (Windows, LabVIEW Real-Time)

You must open TCP ports in addition to UDP ports to configure firewalls and Network Address Translating (NAT) routers to transmit network-published shared variables. Starting at TCP port 59110, open one TCP port for each application you run. By default, the NI-PSP protocol begins looking for available TCP ports at port 59110 and increments upward until it finds an available port for each running application. You manually can configure the range of TCP ports the NI-PSP protocol uses by creating and editing a `LogosXT.ini` file. For LabVIEW Real-Time targets, you can configure the range of TCP ports in the `ni-rt.ini` configuration file.

Resolving Project Conflicts

The LabVIEW project contains conflicts when two or more items with the same qualified name from different paths exist in the project. LabVIEW automatically tracks the hierarchy of every item you include in the project by scanning the VI hierarchy. A yellow warning triangle appears on any conflicting items. The **Project Explorer** window provides several options for resolving conflicts.

- Click the **Resolve Conflicts** button in the **Project Explorer** window to display the **Resolve Project Conflicts** dialog box. Use this dialog box to resolve conflicts by renaming or redirecting the conflicting items to call dependent items from the correct path. If the project does not contain any conflicts, LabVIEW disables the **Resolve Conflicts** button.
- From the **Project Explorer** window, select **Project»Show Item Paths** to display the **Paths** column in the **Project Explorer** window and view the file paths that correspond to the project items. You also can right-click the project root and select **View»Full Paths** from the shortcut menu to display the **Paths** column.
- The **Project Explorer** window includes two pages. The **Items** page displays the contents of the project. The **Files** page displays the project items that have a corresponding file on disk. You can organize

filenames and folders on the **Files** page and perform disk operations such as rename, reorganize, and remove. For example, if you rename a file on the **Files** page, LabVIEW renames that file on disk. You can switch from one page to the other by right-clicking a folder or item under a target and selecting **Show in Items View** or **Show in Files View** from the shortcut menu. The **Show in Items View** and **Show in Files View** shortcut menu items only appear if the item exists in the other view. For example, project items that you have not saved do not have a corresponding file on disk so LabVIEW cannot show the item in the file view.

- When you open a VI from a project, LabVIEW adds all members of the VI hierarchy that are not already in the project to **Dependencies**. LabVIEW organizes **Dependencies** in three folders: `vi.lib`, `user.lib`, and `Items in Memory`. When you open a VI that is not currently in the project, LabVIEW adds the VI to `Items in Memory`. These items remain in the project as long as the VI they reference is in memory. **Dependencies** updates every time you add, remove, or save an item in the project.

Using Folders in the LabVIEW Project

You can add two types of folders to a LabVIEW project, virtual folders and auto-populating folders. Virtual folders organize project items in the **Project Explorer** window and have no corresponding value on disk. On the **Items** page, right-click a target and select **Add»Folder (Snapshot)** from the shortcut menu to add a virtual folder to the project. Auto-populating folders automatically populate and update to reflect the contents of a folder on disk. On the **Items** page, right-click a target or folder and select **Add»Folder (Auto-populating)** from the shortcut menu to add an auto-populating folder to the project. Contents of auto-populating folders do not always match disk contents exactly in the case of project libraries. The project displays project library (`.lvlib`) contents by library hierarchy and not by disk organization. For example, if a VI is a member of the project library, the VI appears in the project under the library file. The VI does not appear under the auto-populating folder.

New Project Dialog Boxes

LabVIEW 8.5 includes the following new LabVIEW project dialog boxes:

- The **Resolve Load Conflict** dialog box appears when LabVIEW loads a file whose dependencies conflict with other items in the project and LabVIEW cannot determine which dependent item to load. Use this dialog box to choose which dependent file to load. You must resolve any conflicts before the file loads.
- The **Hierarchy Conflicts with Project** dialog box appears if you attempt to open a file that conflicts with items already in the LabVIEW

project. LabVIEW cannot open the file because items in the VI hierarchy have the same qualified name as items in the project or **Dependencies**. Use this dialog box to resolve conflicting items between two VI hierarchies in the project.

- The **Add to Project and Update Dependencies** dialog box appears when you open the block diagram of a VI in the LabVIEW project and add a new subVI to the block diagram from disk that has the same qualified name as an item already in the project. Use this dialog box to add a new subVI to the project and update caller items to refer to the correct path. You can add an item to the project even if the item has conflicts with other items in the project. You must resolve the conflicts before opening the item.
- The **Project Dependency Conflicts Detected** dialog box appears when you attempt to add a file to the project that may cause conflicts within the project. Use this dialog box to view new dependencies with conflicts that LabVIEW adds to the project. LabVIEW adds all members of the VI hierarchy that are not already in the project to **Dependencies**. You must resolve the conflicts before opening the item.
- The **Undo File Rename** dialog box appears when you attempt to rename a file on disk that exists in an auto-populating folder in the LabVIEW project. LabVIEW detects the rename on disk and cannot rename the item and redirect dependent items. You must revert the rename on disk or accept the file as a new file in the project and resolve conflicts that might occur. This dialog appears only if the file you attempt to rename is already open in LabVIEW or if the file has callers in the project. Use this dialog box to revert the rename.
- The **Find Callers** dialog box appears when you right-click an item in the **Project Explorer** window and select **Find»Callers** from the shortcut menu. Use this dialog box to find all callers of a specific item in the project. If the item only has one caller, LabVIEW highlights the caller in the **Project Explorer** window.
- The **Find SubVIs** dialog box appears when you right-click an item in the **Project Explorer** window and select **Find»SubVIs** from the shortcut menu. Use this dialog box to find all subVIs of a specific item in the project. If the item only has one subVI, LabVIEW highlights the subVI in the **Project Explorer** window.
- The **Find Items with No Callers** dialog box appears when you right-click the project root in the **Project Explorer** window and select **Find Items with No Callers** from the shortcut menu. Use this dialog box to find all top-level project items with no callers.
- The **Find Conflicts** dialog box appears when you right-click a conflicting item in the **Project Explorer** window and select **Find» Conflicts** from the shortcut menu. Use this dialog box to find all items in the project that conflict with the item you select. If the item only has

one conflict, LabVIEW highlights the conflict in the **Project Explorer** window.

Miscellaneous Project Enhancements

LabVIEW 8.5 includes the following miscellaneous project enhancements:

- `.lvlps` files store project settings that are specific to the local machine. You should not check `.lvlps` files into source control because `.lvlps` files contain settings specific to the computer you use. For example, `.lvlps` files contain the local source code control configuration. LabVIEW saves the `.lvlps` file when you save a project, and correctly renames the file when you rename a project. You can remove or delete `.lvlps` files without affecting the performance or behavior of a project because `.lvlps` files only contain project settings specific to the local machine. If you build an application, LabVIEW does not copy the `.lvlps` file into the built application.
- Search for items in a LabVIEW project using the **Find Project Items** dialog box. Use this dialog box to find targets, VIs, subVIs, libraries, classes and so on. Select **Edit»Find Project Items** from the **Project Explorer** window to access the **Find Project Items** dialog box. Right-click a folder, project library, LabVIEW class, or XControl in a LabVIEW project and select **Find Project Items** from the shortcut menu to search for items within a specific project item. You also can access the **Find Project Items** dialog box from a stand-alone project library window or class window.
- Use the **Project Explorer** window to add a hyperlink as an item in the LabVIEW project. Right-click a target, or a folder or library under the target, and select **Add»Hyperlink** from the shortcut menu to display the **Hyperlink Properties** dialog box. You can use hyperlinks to link to files or directories that are not on the local computer but are accessible on the Internet or your local network. The hyperlinks can correspond to network, local, HTTP, FTP, mailto addresses and so on.
- Rename a LabVIEW project library, XControl, or LabVIEW class by right-clicking the project library, XControl, or LabVIEW class and selecting **Rename** from the shortcut menu to enter the new name.
- The **Save All** button on the toolbar of the **Project Explorer** window changed to the **Save All (this Project)** button. Use the **Save All (this Project)** button to save all files in the current open project.
- Select **File»Save As** on a previously saved LabVIEW project to display the new **Save As** dialog box.

New and Changed VI, Function, and Node Enhancements

LabVIEW 8.5 includes the following new and changed VIs and functions. Refer to the **VI and Function Reference** book on the **Contents** tab of the *LabVIEW Help* for more information about VIs, functions, and nodes.

New VIs and Functions

LabVIEW 8.5 includes the following new VIs and functions.

Cluster, Class, & Variant VI and Functions

The **Cluster, Class, & Variant** palette includes the following new VI and constant.

- Get LV Class Default Value
- LV Object

Geometries VIs

The **Geometries** palette includes the Create Text VI in the LabVIEW Full and Professional Development Systems.

Mathematics VIs

The **Mathematics** palette includes the following new VIs in the LabVIEW Full and Professional Development Systems:

- Constrained Nonlinear Curve Fit
- Create Polynomial From PFE
- Generalized SVD Decomposition
- Nonlinear curve fit intervals
- Uneven Numeric Integration
- Vector Norm

The new **Basic Linear Algebra Subroutines** palette includes the following new VIs in the Full and Professional Development Systems:

- amax - Max Element Index
- amin - Min Element Index
- asum - Absolute Values Sum
- axpy - Scalar-Vector Product
- copy - Vector Copy
- ddot - Dot Product (DBL)
- dger - General Matrix Rank-1 Update (DBL)
- drotm - Fast Givens Rotation (DBL)

- drotmg - Fast Givens Rotation Parameters (DBL)
- dsymv - Symmetric Matrix-Vector Product (DBL)
- dsyr - Symmetric Matrix Rank-1 Update (DBL)
- dsyr2 - Symmetric Matrix Rank-2 Update (DBL)
- gemm - General Matrix-Matrix Product
- gemv - General Matrix-Vector Product
- nrm2 - Vector-2 Norm
- rot - Givens Rotation
- rotg - Givens Rotation Parameters
- swap - Vector Swap
- symm - Symmetric Matrix-Matrix Product
- syr2k - Symmetric Matrix Rank-2k Update
- syrk - Symmetric Matrix Rank-k Update
- trmm - Triangle Matrix-Matrix Product
- trmv - Triangle Matrix-Vector Product
- trsm - Solve Linear Eqs (Triangle, multiple)
- trsv - Solve Linear Eqs (Triangle, single)
- zdotc - Dot Product with Conjugation (CDB)
- zdotu - Dot Product (CDB)
- zgerc - General Matrix Rank-1 Update with Conjugation (CDB)
- zgeru - General Matrix Rank-1 Update (CDB)
- zhemm - Hermitian Matrix-Matrix Product (CDB)
- zhemv - Hermitian Matrix-Vector Product (CDB)
- zher - Hermitian Matrix Rank-1 Update (CDB)
- zher2 - Hermitian Matrix Rank-2 Update (CDB)
- zher2k - Hermitian Matrix Rank-2k Update (CDB)
- zherk - Hermitian Matrix Rank-k Update (CDB)

VISA 64-bit Primitive Functions

The **Register Access** palette includes the following new functions:

- VISA In 64
- VISA Memory Allocation Ex
- VISA Move In 64
- VISA Move Out 64
- VISA Out 64

The **Low Level Register Access** palette includes the following new functions:

- VISA Peek 64
- VISA Poke 64

Improving Memory Allocation and VI Efficiency with the In Place Element Structure

Many common operations, such as operating on an element of an array and placing the resulting value back into the same array index, require LabVIEW to copy data values and maintain those values in memory, which increases memory usage.

Use the In Place Element structure to perform common LabVIEW operations in the same memory location and without LabVIEW making multiple copies of the data values in memory. The In Place Element structure includes a set of border nodes, or nodes that are attached to the border of the In Place Element structure, to perform operations on data within the same memory location. Right-click the border of the In Place Element structure and select the appropriate border node for the data.

You can use the In Place Element structure to index an array, unbundle a cluster or waveform, operate on variant data, or operate on any data type in the same memory location.

Refer to the *In Place Element Structure* topic in the **Fundamentals»Loops and Structures»Concepts** book on the **Contents** tab in the *LabVIEW Help* for more information about the In Place Element structure.

Changed VIs, Functions, and Nodes

The following VIs, functions, and nodes changed in LabVIEW 8.5.

File I/O VIs and Functions

The **File I/O** palette includes the following changed VIs:

- **Close Config Data**—Includes a **file path** output that returns the path to the configuration file.
- **Close Data Storage**—Includes a **file path** output that returns corresponding path to the **storage refnum** input.

Mathematics VIs

The **Mathematics** palette includes the following changed VIs:

- **Add Polynomials**—Both instances include a **threshold** input that specifies the level at which the VI removes the trailing elements and a

threshold type input that specifies how the VI removes the trailing elements.

- **Add Rational Polynomials**—Both instances include a **threshold** input that specifies the level at which the VI removes the trailing elements and a **threshold type** input that specifies how the VI removes the trailing elements.
- **Back Transform Eigenvectors**—The **job** input is now recommended rather than required.
- **Derivative x(t)**—Includes a **method** input that specifies the differentiation method.
- **Divide Polynomials**—Both instances include a **threshold** input that specifies the level at which the VI removes the trailing elements and a **threshold type** input that specifies how the VI removes the trailing elements.
- **Divide Rational Polynomials**—Both instances include a **threshold** input that specifies the level at which the VI removes the trailing elements and a **threshold type** input that specifies how the VI removes the trailing elements.
- **Elliptic Integral of the 1st kind**—Is a polymorphic VI with the following instances: Complete Elliptic Integral K and Incomplete Elliptic Integral F.
- **Elliptic Integral of the 2nd kind**—Is a polymorphic VI with the following instances: Complete Elliptic Integral E and Incomplete Elliptic Integral E.
- **General Polynomial Fit**—This VI now accepts **polynomial order** less than or equal to 25. If **polynomial order** is greater than 25, the VI sets the coefficients in **Polynomial Coefficients** to zero whose orders are greater than 25 and returns a warning.
- **(Incomplete) Beta Function**—Is a polymorphic VI with the following instances: Beta Function and Incomplete Beta Function.
- **(Incomplete) Gamma Function**—Is a polymorphic VI with the following instances: Gamma Function and Incomplete Gamma Function.
- **Lyapunov Equations**—Both instances include an **equation type** input that specifies the type of Lyapunov equation.
- **Multiply Polynomials**—Both instances include a **threshold** input that specifies the level at which the VI removes the trailing elements and a **threshold type** input that specifies how the VI removes the trailing elements.
- **Multiply Rational Polynomials**—Both instances include a **threshold** input that specifies the level at which the VI removes the trailing

elements and a **threshold type** input that specifies how the VI removes the trailing elements.

- **Polynomials Composition**—Both instances include a **threshold** input that specifies the level at which the VI removes the trailing elements and a **threshold type** input that specifies how the VI removes the trailing elements.
- **Quadrature**—Is a polymorphic VI with the following instances: 1D Quadrature (VI) and 1D Quadrature (Formula).
- **Remove Zero Coefficients**—Both instances include a **threshold type** input that specifies how the VI removes the trailing elements.
- **RMS**—Is a polymorphic VI with the following instances: RMS (DBL) and RMS (CDB).
- **Subtract Polynomials**—Both instances include a **threshold** input that specifies the level at which the VI removes the trailing elements and a **threshold type** input that specifies how the VI removes the trailing elements.
- **Subtract Rational Polynomials**—Both instances include a **threshold** input that specifies the level at which the VI removes the trailing elements and a **threshold type** input that specifies how the VI removes the trailing elements.

Pipes VIs

(Linux) The **Pipes** palette includes the following changed VIs:

- **Close Pipe**—The **file descriptor** input changed from recommended to required.
- **Open Pipe**—The **path to named pipe** input changed from recommended to required.
- **Open System Command Pipe**—The **command line** input changed from recommended to required.
- **Read From Pipe**—The **file descriptor** input changed from recommended to required.
- **Write To Pipe**—The **file descriptor** input and the **data** input changed from recommended to required.

Protocol VIs and Functions (Windows and Linux)

The **Protocols** palette includes the following changed VI and functions:

- **TCP Open Connection**—Includes a remote port or service name input. You can provide a service name and LabVIEW queries the NI Service Locator for the port number associated with that service name.

- **TCP Create Listener**—Includes a service name input that registers the service name and the provided port number with the NI Service Locator.
- **TCP Listen**—includes a service name input that registers the service name and the provided port number with the NI Service Locator.
- **UDP Open**—Includes a service name input that registers the service name and the provided port number with the NI Service Locator.
- **UDP Write**—Includes a port or service name input. You can provide a service name and LabVIEW queries the NI Service Locator for the port number associated with that service name.

Signal Processing VIs

The **Signal Processing** palette includes the following changed VIs:

- **Ramp Pattern**—Includes a **type** input that specifies the type of **Ramp Pattern** to generate.
- **Transition Measurements**—Both instances include the following changes:
  - The **slew rate** output changed to **slope**.
  - The **duration** output changed to **transition duration**.
  - The **preshoot** output changed to **pre-transition**. This output includes an **undershoot** element that measures the height of the local minimum preceding a rising (falling) transition as a percentage of the histogram-based amplitude of the signal. The **pre-transition** output also includes an **overshoot** element that measures the height of the local maximum preceding a rising (falling) transition as a percentage of the histogram-based amplitude of the signal.
  - The **overshoot** output changed to **post-transition**. This output includes an **undershoot** element that measures the height of the local minimum following a rising (falling) transition as a percentage of the histogram-based amplitude of the signal. The **post-transition** output also includes an **overshoot** element that measures the height of the local maximum following a rising (falling) transition as a percentage of the histogram-based amplitude of the signal.
- **Unwrap Phase**—Includes a **phase unit** input that specifies the units for **Phase** and **Unwrapped Phase**.
- **Window Properties**—Is a polymorphic VI with the following instances: Window Properties by Coef and Window Properties by Name. The Window Properties by Name instance of this polymorphic VI includes the inputs **size**, **window**, and **window parameter**.

Waveform VIs and Functions

The **Waveform Monitoring** palette includes the following changed VI.

On the Basic Level Trigger Detection, the **mode** input is renamed **location mode**.

Cluster, Class & Variant VI and Functions Palette

The **Cluster & Variant Functions** palette is renamed to **Cluster, Class, & Variant VI and Functions** palette. The subpalette **Variant Attributes Functions** is renamed **Variant Functions**. Several objects on each palette have moved.

The **Cluster, Class, & Variant VI and Functions** palette includes the following VI and functions:

- Array To Cluster
- Build Cluster Array
- Bundle
- Bundle By Name
- Call Parent Method
- Cluster
- Cluster to Array
- Get LV Class Default Value
- Index & Bundle Cluster Array
- LV Object
- To More Generic Class
- To More Specific Class
- Unbundle
- Unbundle By Name

The **Variant Functions** subpalette includes the following functions:

- Delete Variant Attribute
- Flattened String To Variant
- Get Variant Attribute
- Set Variant Attribute
- To Variant function
- Variant To Data
- Variant To Flattened String

Adding a Conditional Terminal to a For Loop

You can add a conditional terminal to configure a For Loop to stop early when a Boolean condition or error occurs. A For Loop with a conditional terminal executes until a Boolean condition or an error occurs or until all iterations complete, whichever happens first. To add a conditional terminal to a For Loop, right-click the loop border and select **Conditional Terminal** from the shortcut menu.

Refer to the **Fundamentals»Loops and Structures** book on the **Contents** tab in the *LabVIEW Help* for more information about For Loops in LabVIEW.

Using Feedback Nodes to Store Data

Use a Feedback Node anywhere on the block diagram to store data from a previous VI or loop execution. When you place a Feedback Node on the block diagram, LabVIEW automatically attaches an initializer terminal to it.

If you place a Feedback Node in a loop, you can move the initializer terminal to the left edge of the loop by clicking and dragging the terminal or by right-clicking the Feedback Node and selecting **Move Initializer One Loop Out**. When the initializer terminal is on the left edge of a loop, you can select **Move Initializer One Loop In** or **Globally Initialize** from the shortcut menu to move it back to the node. If you move the initializer terminal to the left edge of a loop, the Feedback Node initializes before the loop with the initializer terminal begins the first iteration of each execution. If you select **Globally Initialize** and wire an input value to the initializer terminal, the Feedback Node initializes on the first call of the VI in an execution. The initializer terminal remains attached to the node and the Feedback Node globally initializes by default.

If you choose to save a VI for a previous version of LabVIEW, LabVIEW replaces Feedback Nodes within loops with shift registers and replaces Feedback Nodes outside of loops with labels.

Refer to the **Fundamentals»Local Variables, Global Variables, and the Feedback Node** and **Fundamentals»Loops and Structures** books on the **Contents** tab in the *LabVIEW Help* for more information about Feedback Nodes in LabVIEW.

Miscellaneous VI, Function, and Node Changes

LabVIEW 8.5 includes the following miscellaneous VI, function, and node changes:

- The Unzip VI unzips the contents of a zip file that is not password protected to the directory you specify.

- The Sound File Write VI includes the following instance: DBL Single. For monophonic sound data, you can wire a single waveform data type to the **data** input of this instance.
- **(Windows and Linux)** The **port number** input of the Open Application Reference function changed to **port number or service name**. You can provide a service name and LabVIEW queries the NI Service Locator for the port number associated with that service name.
- Performance of VIs on the **Probability & Statistics**, **Windows**, **Transforms**, and **Signal Generation** palettes is improved.
- The **reference** input and the **reference out** output of the Call Library Function Node changed to **path in** and **path out**, respectively.
- The Preview Queue Element function icon changed.
- You can use `NI_UpdateGroupName` and `NI_UpdateChannelName` properties in the TDMS Set Properties function to rename groups and channels in `.tdms` files.
- Icons for VIs on the **Input Device Control** palette are updated.
- The Spectral Measurements Express VI accepts the waveform array of waveforms data types as an input.
- The Formula Express VI accepts the numeric scalar, 1D array of numerics, 2D array of numerics, waveform, and array of waveforms data types as an input.
- **(Mac OS and Linux)** The Copy and Move functions no longer overwrite read-only files or folders in the target path, even if you set **overwrite** to TRUE.
- There are new names and values for certain **Math & Scientific Constants** and **Express Math & Scientific Constants**. The values for Planck's Constant, Elementary Charge, Gravitational Constant, Avogadro Constant, and Rydberg Constant are updated to match values provided by CODATA 2002.
- The VISA Wait on Event function and the Wait for RQS VI no longer lock out other GPIB operations while they are executing.
- The Number to Boolean Array function now accepts 64-bit integers and the fixed-point data type as inputs.
- The LabVIEW Class:Create method of the Application class includes a name parameter you can use to specify the name of the LabVIEW class.
- The Mass Compile and MassCompile methods of the Application class and Application (ActiveX) classes, respectively, include the User Stopped parameter that indicates whether the user stopped the mass compile operation.
- Because LabVIEW may generate multiple Pane Size events when you resize a pane, LabVIEW now generates a final event at the end of the

resizing operation in which the **OldBnds** and **NewBnds** parameters return the same value. You can use the values to identify in the code when the sizing operation finishes.

- The Request Deallocation function is now located on the **Memory Control** palette.
- The Close Reference function now accepts the 1D array of refnum data type as an input.
- When using arrays of numbers as inputs for the Formula Express VI, the length of the output array is now the same length as the shortest input array.
- The Type Cast function accepts 64-bit integers and double-precision floating point numbers.

New Classes, Properties, Methods, and Events

LabVIEW 8.5 includes new VI Server classes, properties and methods and events. Refer to the **LabVIEW 8.5 Features and Changes»New VI Server Objects** topic on the **Contents** tab of the *LabVIEW Help* for a list of new class, properties, methods, and events.

Application Builder Enhancements

LabVIEW 8.5 includes the following enhancements to the Application Builder.

In LabVIEW 8.5, the **Source Distribution Properties** dialog box contains most of the same pages as the **Application** and **Shared Library Properties** dialog boxes. The **Application Properties**, **Shared Library Properties**, and **Source Distribution Properties** dialog boxes have many of the same changes. The changes below apply to all three dialog boxes unless otherwise noted.

Information Page

LabVIEW 8.5 includes the following changes to the **Information** page:

- The **Application destination directory**, **Shared Library destination directory**, and **Distribution destination directory** text boxes changed to **Destination directory**.
- Use the **Build specification description** text box to specify a description for the build specification.
- **(Source Distribution)** The **Directory** and **Preserve hierarchy** options on the **Destinations** page replace the **Packaging Option** functionality.

Source Files

LabVIEW 8.5 includes the following changes to the **Source Files** page:

- The **Dynamic VIs and Support Files** listbox changed to **Always Included**.
- (**Source Distribution**) Use the **Always Excluded** listbox to designate files that should be excluded from the build.
- The **Source Distribution Properties** dialog box includes the **Source Files** page. You must indicate which files you want to include in a source distribution. LabVIEW no longer automatically includes all files in the LabVIEW project.
- You can right-click and select multiple files in the **Project Files** tree to add several files at one time to the other listboxes.
- The **Destination View** list includes the following new locations:
  - `[CommonAppDataFolder]`—Files you include in this folder install to the `Application Data` folder.
  - `[CommonFilesFolder]`—Files you include in this folder install to the `Common Files` folder.

Destinations Page

LabVIEW 8.5 includes the following changes to the **Destinations** page:

- **Destination is LLB** changed to **LLB** in the **Destination type** section.
- Use the **Directory** and **Preserve hierarchy** options to indicate that the build destination is a directory and whether you want to preserve the directory hierarchy. You can use these options with custom destinations. Additionally, these options replace the **Packaging Option** on the former **Distribution Settings** page of the **Source Distribution Properties** dialog box.
- Use the **Add files to new project library** option to add files to a new project library created during the build process. Designate the project library name in the **Library name** textbox.
- The **Source Distribution Properties** dialog box includes the **Destinations** page.

Source Files Settings Page

LabVIEW 8.5 includes the following changes to the **Source Files Settings** page:

- The **Inclusion type** section is an indicator and reflects the value you selected on the **Source Files** page. To change the value, return to the **Source Files** page and move the selected item to a different listbox.

- The **Destination** indicator is a pull-down menu. You can select a different destination directory from the pull-down menu.
- The **VI Settings** section moved to the **VI Properties** dialog box. You can access this dialog box by clicking the **Customize VI Properties** button. The options in the **VI Properties** dialog box correspond with the options in the **VI Properties** dialog box that you access from the **File** menu. If you make changes to the VI properties in this dialog box, those changes override any changes you made in the **VI Properties** dialog box menu from the **File** menu.
- Use the options in the **Use default save settings** section to remove the block diagram or front panel of the selected item.
- Use the **No password change**, **Remove password**, and **Apply new password** options to configure the password settings for the file selected in the **Project Files** tree.
- Use the **Rename this file in the build** option to rename the file selected in the **Project Files** tree during the build process. If you choose to rename a folder in the build, LabVIEW adds a prefix to the items in the selected folder.
- If you select a folder in the **Project Files** tree, you can choose to apply many of the options on the **Source Files Settings** page to all items in the folder.

Icon Page (Windows, Mac OS)

(Application) Use the **Icon Image** pull-down menu to display the resolution and color options for an icon image. The image preview displays the resolution and color options the image supports, up to 256 × 256 and 32-bit color.

Advanced Page

LabVIEW 8.5 includes the following changes to the **Advanced** page:

- The **Enable Mathscript support** checkbox no longer exists. LabVIEW automatically detects Mathscript calls to user-defined functions, or `.m` files.
- Several options have changed location on this page to improve ease of use.

Additional Exclusions Page

LabVIEW 8.5 includes the following changes to the **Additional Exclusions** page:

- **Disconnect type definitions**—Disconnects type definitions from the build.

- **Remove unused polymorphic VI instances**—Removes all unused polymorphic VI instances from the build.
- **Remove unused members of project libraries**—Removes unused members of project libraries from the build.
  - **Modify project library file after removing unused members**—Modifies the library file so the library file does not reference the removed members.

The following table displays the LabVIEW 8.5 options, noted previously, that you can select to achieve the same exclusions as LabVIEW 8.2.

LabVIEW 8.2 Option	LabVIEW 8.5 Option
Remove as much as possible	Select all three new options.
Remove unreferenced project library members	Select **Remove unused members of project libraries**. You also can select **Modify project library file after removing unused members** so the library file does not reference the removed members.
Do not disconnect type definitions or remove unreferenced members	Do not select any option.

Version Information Page (Windows, Mac OS)

The version information on the **Application** and **Shared Library Properties** dialog box pages now appears on the **Version Information** page. The build specification name populates the **Description** by default. Users can view this information by right-clicking the built application and selecting **Properties** from the shortcut menu.

Run-Time Languages

You can select the supported languages in the **Supported Languages** listbox by enabling checkboxes next to each language.

Miscellaneous Application Builder Enhancements

- In the **Define VI Prototype** dialog box, the **C Calling Conventions** radio button is enabled by default.
- Right-click **Build Specifications** in the **Project Explorer** window and select **Build All** to build all specifications under **Build Specifications**.
- If you convert a build script from a previous version of LabVIEW to a build specification, LabVIEW places the items for the build specification in three folders in the **Project Explorer** window rather than creating a hierarchy.

- The **Zip File Properties** dialog box includes the **Zip File Structure** page. Use this page to specify the file structure to use for the zip file build and, optionally, to rename the base directory to which the files unzip.
- You can include outputs of other build specifications, such as source distributions, shared libraries, or applications, in a zip file you build using the **Zip File Properties** dialog box. On the **Source Files** page, select the build specification in the **Project Files** tree and use the right arrow button to add it to the **Included Items** tree.

  Outputs of zip or installer build specifications do not appear in the **Project Files** tree. To include another zip file or installer in the zip file, add the zip file or installer you want to include to the LabVIEW project under the target from which you are building the zip file.

Note Including outputs of other build specifications in a zip file can slow down preview generation on the **Preview** page.

LabVIEW MathScript Enhancements (Windows, Not in Base Package)

LabVIEW 8.5 introduces the following enhancements and changes to MathScript.

Note Select **Tools»MathScript Window** to display the **LabVIEW MathScript Window**.

New MathScript Functions

LabVIEW 8.5 includes the following new MathScript functions. You can use these functions in the **LabVIEW MathScript Window** or the MathScript Node.

- `advanced` class: `bessel`
- `approximation` class: `interpft`
- `audio` class: `soundsc` and `wavrecord`
- `basic` class: `cumtrapz`, `feval`, `ipermute`, and `permute`
- `dsp` class: `cohere`, `csd`, `firpmord`, `psd`, and `tfe`
- `geometry` class: `cylinder`, `ellipsoid`, `inpolygon`, `rectint`, `sphere`, and `voronoi`
- `integration` class: `dblquad`, `quad8`, `quadl`, and `triplequad`
- `linalgebra` class: `eigs`, `funm`, `gsvd`, and `linsolve`
- `membership` class: `isscalar`
- `plots` class: `axes`, `colormap`, `datetick`, `gca`, `get`, `ginput`, `gtext`, `image`, `pareto`, `rgbplot`, `set`, `surfnorm`, and `view`

- `polynomials` class: `pchip` and `polyder`
- `statistics` class: `halton` and `richtmeyer`
- `string` class: `evalc`, `regexp`, `regexpi`, `regexprep`, `regexptranslate`, `strread`, and `strtrim`
- `support` class: `delete`, `dos`, `feof`, `fgetl`, `fgets`, `frewind`, `fseek`, `getfileproperty`, `imread`, `imwrite`, `lookfor`, `setfileproperty`, `system`, and `textread`
- `vector` class: `del2` and `divergence`
- `zerofinder` class: `fsolve` and `fzero`

Enhanced Error Reporting for the MathScript Node

LabVIEW 8.5 includes enhanced error reporting for the MathScript Node. For example, if you call a user-defined function, or `.m` file, from a MathScript Node, LabVIEW returns edit-time errors in the user-defined function at edit time instead of at run time. If you make changes to the user-defined function, LabVIEW updates the **Error list** window as soon as you save the `.m` file. However, if you call a function from a MathScript Node that might change the MathScript search path list or might introduce new variables at run time, LabVIEW operates with reduced error checking and slower run-time performance for the MathScript Node.

A warning glyph appears on the MathScript Node frame to indicate the reduced error checking and slower run-time performance. The following functions cause the warning glyph to appear: `addpath`, `cd`, `clear`, `eval`, evalc, `load`, `path`, `rmpath`, and `uiload`. The `cd` function and the `path` function cause the warning glyph to appear only if you call these functions with one or more inputs. The warning glyph also appears if you call a user-defined function that calls any of these functions, if you call a user-defined script, or if you call a user-defined function that calls a user-defined script. To remove the warning glyph from the MathScript Node and improve run-time performance, complete the following steps.

1. Remove these functions from scripts and user-defined functions.
2. Remove calls to user-defined scripts. Instead, copy the contents of the user-defined script into the MathScript Node or the user-defined function that calls the script.
3. Do not change the MathScript search path list at run time. Instead, use the **MathScript: Search Paths Options** page to configure the default search path list.

Note If a script in a MathScript Node calls a user-defined function, LabVIEW uses the default search path list to link the function call to the specified `.m` file. After you configure the default search path list and save the VI that contains the MathScript Node, you do not need to reconfigure the MathScript search path list when you open the VI on a different computer because LabVIEW looks for the `.m` file in the directory where the `.m` file was located when you last saved the VI. However, you must maintain the same relative path between the VI and the `.m` file.

Calling User-Defined Functions from LabVIEW MathScript

When you call a user-defined function from the **LabVIEW MathScript Window**, LabVIEW searches the MathScript search path list from top to bottom for a `.m` file with the specified name. When you call a user-defined function from a MathScript Node, LabVIEW looks in the following three places, in order, for a `.m` file with the specified name:

- First, LabVIEW looks to see if a `.m` file with the specified name is currently in memory.
- Second, LabVIEW looks in the directory where the `.m` file was located when you last saved the VI that contains the MathScript Node. Because LabVIEW looks in the directory where the `.m` file was last located, you do not need to reconfigure the MathScript search path list when you open the VI on a different computer. However, you must maintain the same relative path between the VI and the `.m` file.
- Third, LabVIEW searches the MathScript search path list from top to bottom. If you open the VI on a different computer, the search path list might change because LabVIEW searches the MathScript search path list configured on that computer, not the search path list configured when you last saved the VI.

Note If a warning glyph appears on the MathScript Node frame, LabVIEW does not look in memory or in the directory where the `.m` file was last located. Instead, LabVIEW searches only the MathScript search path list for a `.m` file with the specified name.

Configuring the MathScript Search Path

Use the **MathScript: Search Paths Options** page to configure the default search path list and to specify the working directory for LabVIEW MathScript. Display this page from one of the following locations, depending on the environment for which you want to configure the search path list:

- To configure the search path list for the **LabVIEW MathScript Window**, display this page from the **LabVIEW MathScript Properties** dialog box.
- To configure the search path list for MathScript Nodes in the main application instance, display this page from the **Options** dialog box.
- To configure the search path list for MathScript Nodes in a target on a LabVIEW project, display this page from the **Properties** dialog box for the target.

Using the LabVIEW MathScript Probe

Use the LabVIEW MathScript probe to view the data in a script in a MathScript Node as a VI runs. Right-click a MathScript Node and select **Probe** from the shortcut menu to use the MathScript probe.

The MathScript probe displays a list of all variables you define in the script and previews variables you select. The MathScript probe also displays the output that MathScript generates from the script. Unlike with supplied probes, you cannot configure the MathScript probe to respond to the data. For example, you cannot specify that you want to set a conditional breakpoint.

Note You cannot use execution highlighting, single-stepping, and breakpoints within a script in a MathScript Node.

Setting Plot Attributes in LabVIEW MathScript

You can get and set plot attributes for the following types of objects in LabVIEW MathScript plots: line objects, plot area objects, plot window objects, and text objects.

To create and return a new line object, use the following MathScript functions: `line`, `loglog`, `plot`, `semilogx`, or `semilogy`. To create a new plot area object or to return an object for the current plot area, use the following MathScript functions: `axes`, `gca`, or `subplot`. To create a new plot window object or to return an object for the current plot window, use the following MathScript functions: `clf`, `gcf`, or `figure`. To create and return a new text object, use the following MathScript functions: `text`, `title`, `xlabel`, or `ylabel`.

Dialog Box Enhancements

LabVIEW 8.5 includes the following MathScript dialog box enhancements.

New MathScript Dialog Box

LabVIEW 8.5 includes the **LabVIEW MathScript Properties** dialog box. In the **LabVIEW MathScript Window**, select **File»LabVIEW MathScript Properties** to display this dialog box. Use this dialog box to configure settings for the **LabVIEW MathScript Window** and the MathScript engine. This dialog box includes the following pages:

- **MathScript: Window Options**—Use this page to configure settings, such as the font style and the display format of numbers, for the **LabVIEW MathScript Window**.

- **MathScript: Search Paths Options**—Use this page to configure the default search path list and to specify the working directory for LabVIEW MathScript.

Miscellaneous Dialog Box Enhancements

LabVIEW 8.5 includes the following miscellaneous enhancements to MathScript dialog boxes:

- The **MathScript Preferences** dialog box no longer exists. Use both the **MathScript: Search Paths Options** page and the **MathScript: Window Options** page of the **LabVIEW MathScript Properties** dialog box instead.
- The **Preferences** dialog box no longer exists. Use the **MathScript: Window Options** page of the **LabVIEW MathScript Properties** dialog box to configure settings that previously existed on the **History** page and the **Output** page.
  - All options in the deprecated **System Info** page no longer exist.
  - The **History buffer size** option in the **History** page changed to **History buffer** and moved to the **MathScript: Window Options** page of the **LabVIEW MathScript Properties** dialog box.
- The **Script Editor** includes a **New Script** button that clears the script in the **Script Editor**.

Menu Item Enhancements

In LabVIEW 8.5, the menus in the **LabVIEW MathScript Window** align more closely with the menus in LabVIEW. The menus in the **LabVIEW MathScript Window** include the following enhancements:

- The **File** menu includes the following changes:
  - The **Save Script** item changed to **Save**.
  - The **Save Script As** item changed to **Save As**.
  - The **Save & Compile Script As** item no longer exists.
- In the **Edit** menu, the **Delete** item no longer exists.
- The **View** menu includes the following changes:
  - The **Workspace** item no longer exists. You cannot hide the **LabVIEW MathScript Window** workspace.
  - The **Output Wrapped** item no longer exists. Place a checkmark in the **Output wrapped** checkbox on the **MathScript: Window Options** page of the **LabVIEW MathScript Properties** dialog box to enable word wrapping in the **Output Window**.

- The **Operate** menu includes a **Run Script** item that executes all commands in the **Script Editor**.
- The **Load Data**, **Save Data**, and **Load Script** items moved from the **File** menu to the **Operate** menu.

Miscellaneous MathScript Enhancements and Changes

LabVIEW 8.5 includes the following miscellaneous changes to MathScript:

- LabVIEW MathScripts that contain matrix indexing operations execute faster at run time due to performance optimizations.
- If you call a user-defined function from a MathScript Node that does not include a warning glyph, LabVIEW executes faster at run time due to performance optimizations to the MathScript Node.
- When you reference a `.m` file from a VI that is part of a LabVIEW project, LabVIEW adds the `.m` file to the dependencies for the target.
- The MathScript Node shows line numbers by default. You can show and hide line numbers in the MathScript Node, the Formula Node, the MATLAB script node, and the Xmath script node. To show or hide line numbers, right-click inside the node and select **Visible Items»Line Numbers** from the shortcut menu.
- You can resize individual sections of the **LabVIEW MathScript Window**, such as the **Command Window** and the **Output Window**. To resize a section, move the splitter bar located between the two sections you want to resize.
- The following functions include an **obj** output that returns the reference to the line object, the plot area object, the plot window object, or the text object: `clf`, `figure`, `gcf`, `line`, `loglog`, `plot`, `semilogx`, `semilogy`, `subplot`, `text`, `title`, `xlabel`, and `ylabel`.
- The following functions include a **name** input that specifies the attribute name and a **value** input that specifies the attribute value: `figure`, `line`, `loglog`, `plot`, `semilogx`, `semilogy`, `text`, `title`, `xlabel`, and `ylabel`.
- The `waitforbuttonpress` function waits for a key up event or a mouse up event before continuing execution. In a plot window, the mouse up event must occur in the plot area or outside of the plot window. If you click the scroll bar, the menu bar, the title bar, or the toolbar, this function does not continue execution.
- The `legend` function includes a **location** input that specifies where to add the legend to the plot. You also can use the `legend off` syntax to hide the legend.

- The `rand` and `randn` functions include an **s** input that specifies the seed of the pseudorandom number generator to use for the next call to the function. These functions also include a **d** output that returns the seed of the pseudorandom number generator.
- The `butter`, `cheby1`, `cheby2`, and `ellip` functions include an **'s'** input that directs LabVIEW to design an analog filter.
- The **fs** input of the `remezord` function changed from required to optional. The default is 2.
- The **b** input of the `zplane` function changed from required to optional. If **a** is a row vector and you do not specify **b**, the pole is (0, 0). If **a** is a column vector or a matrix and you do not specify **b**, LabVIEW does not plot poles.
- The `help` command supports the `help browser` syntax that provides an overview of the **LabVIEW MathScript Window**.
- You can include the following functions in a stand-alone application or shared library: `fopen`, `fclose`, `fread`, and `fwrite`.

Refer to the **Fundamentals»Formulas and Equations** book on the **Contents** tab in the *LabVIEW Help* for more information about LabVIEW MathScript.

LabVIEW Object-Oriented Programming

LabVIEW 8.5 introduces the following enhancements and changes to LabVIEW object-oriented programming.

Accessing LabVIEW Class Data

You can quickly create member VIs that can access the LabVIEW class data using the **Create Accessor** dialog box. You can create static or dynamic accessor VIs to read from or to write to class data. The read accessor VI unbundles LabVIEW class data so you can access the data in a calling VI. The write accessor VI bundles new values for class data that you pass from the calling VI.

To save a significant amount of development time, you can select multiple elements of the class data to create several accessor VIs at one time. You can save additional development time by creating both read and write accessor VIs at one time. Consider using accessor VIs as a starting point to creating more complex VIs.

You must save the LabVIEW class before using this dialog box to create accessor VIs.

You can launch the **Create Accessor** dialog box from the **Project Explorer** window in the following ways:

- Right-click the LabVIEW class and select **New»VI for Data Member Access** from the shortcut menu.
- Right-click a data member in the private data control of the LabVIEW class and select **Create Accessor** from the shortcut menu.

Note You must save a new LabVIEW class before using either of these options. LabVIEW dims the **VI for Data Member Access** and **Create Accessor** options if you have not saved the new class.

Opening Implementations of Dynamic Dispatch SubVIs

Dynamic dispatch subVIs can call any one of a set of VIs in a LabVIEW class hierarchy. LabVIEW determines which implementation of the subVI to call at run time, depending on the class data type flowing into the dynamic dispatch terminal.

Double-click a dynamic dispatch subVI to display the **Choose Implementation** dialog box. You can use this dialog box to view all implementations of a dynamic dispatch subVI that are currently in memory and then open one or more implementations of the subVI.

Shortcut Menu Changes

LabVIEW 8.5 includes the following shortcut menu changes for LabVIEW object-oriented programming features:

- The **New»Dynamic VI** shortcut menu option changed to **New»VI from Dynamic Dispatch Template**. Access this option by right-clicking a LabVIEW class.
- The **New»Override VI** shortcut menu option changed to **New»VI for Override**. Access this option by right-clicking a LabVIEW class.
- Right-click a class constant, control, or indicator on the front panel or block diagram and select **Show Class Library** from the shortcut menu to highlight the class in the **Project Explorer** window. If the class you are working with is not in a LabVIEW project, LabVIEW opens a class window to display the class.

Miscellaneous LabVIEW Object-Oriented Programming Enhancements

LabVIEW 8.5 includes the following miscellaneous LabVIEW object-oriented programming enhancements:

- Use dynamic dispatch member VIs in a LabVIEW class to take advantage of recursion. Recursive VIs can call themselves on their own block diagram including on the block diagram of any subVIs. Recursion is useful if you want to operate many times on the output of the same process. You can configure a member VI to allow recursion by using the **Share clones between instances** option on the **Execution Properties** page of the **VI Properties** dialog box.
- You can create a member VI from a static dispatch template by right-clicking the LabVIEW class and selecting **New»VI from Static Dispatch Template**. LabVIEW populates the new member VI with **error in** and **error out** clusters, a Case structure for error handling, the input LabVIEW class, and the output LabVIEW class. Contrary to creating a dynamic dispatch VI, LabVIEW does not set the input and output terminals as dynamic on the connector pane of the static dispatch VI.
- If you create a new LabVIEW class, LabVIEW prompts you to name the class when you create it.

Refer to the **Fundamentals»LabVIEW Object-Oriented Programming** book on the **Contents** tab in the *LabVIEW Help* for more information about object-oriented programming in LabVIEW.

Project Library and LabVIEW Class Icon Changes

The default icon for a project libraries and LabVIEW classes you create in LabVIEW 8.5 does not contain a number on the icon mask and includes different mask colors that are more compatible with black and white icons. To accommodate the new mask design and to avoid the library or class mask obscuring any part of the VI icon, use an image no larger than 32 pixels wide by 19 pixels high.

You also can edit the library or class icon to use a different mask on the **General Settings** page of the **Project Library Properties** dialog box or **Class Properties** dialog box, respectively.

You can more easily create VI icons for dynamic dispatch VIs you create using the **New»VI for Override** shortcut menu option. The resulting VI retains the ancestor VI icon and overlays the icon of the child class.

Enhancing Virtual Memory Usage

(Windows) LabVIEW 8.5 is large address aware. On a 64-bit operating system, LabVIEW can access up to 4 GB of virtual memory by default. On a 32-bit OS, LabVIEW can access up to 2 GB of virtual memory by default. You can enable LabVIEW to access up to 4 GB of virtual memory on a 32-bit OS by modifying the Windows boot configuration settings. Enabling LabVIEW to access more virtual memory decreases the likelihood of experiencing general LabVIEW errors related to memory allocation when you work with large sets of data.

- **(Windows Vista x64 Edition)** LabVIEW can access up to 4 GB of virtual memory by default.
- **(Windows Vista)** To enable LabVIEW to access up to 3 or 4 GB of virtual memory, open the command line window as an administrator and use `bcdedit` commands to add an entry in the Boot Configuration Data (BCD) store. To open the command line window as an administrator, navigate to the window in the Windows **Start** menu, right-click the program name, and select **Run as administrator** from the shortcut menu.
- **(Windows XP/2000)** To enable LabVIEW to access up to 3 GB of virtual memory, you can add the `/3GB` tag to the Windows `boot.ini` file on the line that specifies the Windows version to boot. If you have more than 4 GB of physical RAM, you can use the `/PAE` tag in place of the `/3GB` tag to enable LabVIEW to access up to 4 GB of virtual memory.

Refer to the *VI Memory Usage* topic in the **Fundamentals»Managing Performance and Memory»Concepts** book on the **Contents** tab in the *LabVIEW Help* for more information about using virtual memory in LabVIEW.

Merging VIs

LabVIEW provides a merge application that you can use to merge differences between an original VI and two revisions of the VI. Use the **Merge VIs** dialog box to merge changes between VIs.

Select **Tools»Merge VIs** from the front panel or block diagram to display the **Merge VIs** dialog box. Specify the original VI in the **Base VI** field. Specify the two VIs to merge in the **Their VI** and **Your VI** fields. Click the **Merge** button to merge the selected VIs.

You also can configure a third-party source control provider to use `LVMerge.exe` as the default merge application. `LVMerge.exe` is the programmatic equivalent to the **Merge VIs** dialog box.

Refer to the **Fundamentals»Development Guidelines»How-To» Merging VIs** book on the **Contents** tab in the *LabVIEW Help* for more information about merging VIs.

Import Shared Library Wizard Enhancements

Use the **Import Shared Library** wizard to create or update a LabVIEW project library of wrapper VIs based on functions in a Windows `.dll` file, a Linux `.so` file, or a Macintosh `.framework` file. The **Import Shared Library** wizard includes the following miscellaneous enhancements:

- **Support for pointers**—You can import functions that contain structures with pointer elements. You also can specify whether to allocate memory for the pointer data before LabVIEW calls the function.
- **Automatic generation of custom controls**—For functions that contain structures, the **Import Shared Library** wizard creates custom controls for elements of the structure, including pointers, automatically.
- **Total project management**—If you select the **Update VIs for a shared library** option on the **Specify Create or Update Mode** page, the wizard retains the most recent settings for each individual function in the shared library. For example, if you have a shared library that contains three functions, you might update only the second function. The next time you run the wizard on that shared library file, it retains the original settings for functions one and three and the new settings for function two.
- **Support for the enhanced Call Library Function Node**—The wrapper VIs utilize Call Library Function Node enhancements from LabVIEW 8.2. The enhancements include support for the array data pointer and array minimum size, and improved error reporting functionality.
- **Function details**—You can select a function on the **Select Functions to Convert** page to view detailed information about that function. The wizard also adds information about the function to the context help of the wrapper VI.
- **Enhanced error reporting**—If the wizard encounters errors or warnings while trying to parse the header file, the warnings and possible solutions appear when you select the function on the **Select Functions to Convert** page. This functionality replaces the **Warnings** page from LabVIEW 8.2.
- **Indicator generation for return values**—You can select whether to create an indicator for return values in a wrapper VI.
- **Cached header parser results**—If you run the wizard on the same shared library multiple times and do not modify the header file, the

wizard caches the results of the header file parsing to improve performance.

- **Configure Include Paths and Preprocessor Definitions Page**—The **Configure Include Paths and Predefined Symbols** page is renamed the **Configure Include Paths and Preprocessor Definitions** page.

Refer to the **Fundamentals»Calling Code Written in Text-Based Programming Languages»Concepts»Importing Shared Libraries** book on the **Contents** tab in the *LabVIEW Help* for more information about the **Import Shared Library** wizard.

3D Picture Control Enhancements

You can add text objects to a 3D scene. The text appears when you generate the 3D scene. Use the Create Object VI and the Create Text VI to add a text object to a 3D scene. You also can use the SceneText properties and methods to configure attributes of the text object, such as color or character size, and to access fonts not installed on the local system.

The Set Drawable method replaces the Set Geometry method for applying a geometry or text to a 3D object. The Set Geometry method is supported in LabVIEW 8.5 for VIs last saved in LabVIEW 8.2, but you cannot wire a text object to this method.

Refer to the **Fundamentals»Graphics and Sound VIs** book on the **Contents** tab in the *LabVIEW Help* for more information about creating 3D scenes in LabVIEW.

Using Timed Structures for Multi-Core Programming

Multi-core programming refers to two or more processors in one computer, each of which can simultaneously run a separate thread. A multithreaded application can have multiple separate threads executing simultaneously on multiple processors. LabVIEW 5.0 was the first version of LabVIEW to include multi-core programming. LabVIEW 5.0 separated the user interface thread from the execution threads. Block diagrams executed in one, or possibly more than one, thread, and front panels updated in another thread. The operating system preemptively multitasked between threads by granting processor time to the execution thread, then to the user interface thread, and so on.

The Timed Loop and Timed Sequence structures include a **Processor** input that allows you to manually assign available processors to handle the execution of the structures. You can configure the processor assignment by wiring an input to the **Processor** input of the Input Node for the structure or for frames of the structure. **(Windows)** You also can configure processors to handle timed structures in the **Processor Assignment** section of the

Configure Timed Loop, **Configure Timed Loop with Frames**, **Configure Next Frame Timing**, and **Configure Next Iteration** dialog boxes.

If you load a VI with a timed structure from a previous version of LabVIEW, the **Processor** input sets to `0` by default, where `0` represents the first available processor in the system, because all timed structures automatically run on processor `0` in previous versions of LabVIEW.

Fixed-Point Data Type

The fixed-point data type is a numeric data type that represents a set of rational numbers using binary digits, or bits. Unlike the floating-point data type, which allows the total number of bits LabVIEW uses to represent numbers to vary, you can configure fixed-point numbers to always use a specific number of bits. Hardware and targets that only can store and process data with a limited or fixed number of bits then can store and process the numbers. You can specify the range and precision of fixed-point numbers.

Use fixed-point representation when you do not need the dynamic functionality of floating-point representation or when you want to work with a target that does not support floating-point arithmetic, such as an FPGA target.

To set a number to fixed-point representation, right-click the numeric object and select **Representation** from the shortcut menu to change the data type of the object. You also can configure the encoding, word length, integer word length, range, and desired delta for fixed-point numbers and specify how **Numeric** functions handle overflow and quantization conditions for fixed-point numbers. To configure a fixed-point number, right-click a constant, control, indicator, or Numeric function and select **Properties** from the shortcut menu to display the **Numeric Properties**, **Numeric Constant Properties**, or **Numeric Node Properties** dialog box.

Refer to the *Numeric Data* topic in the **Fundamentals»Building the Block Diagram»Concepts** book on the **Contents** tab of the *LabVIEW Help* for more information about the fixed-point data type.

Source Control Enhancements

Refer to the **Fundamentals»Organizing and Managing a Project** book on the **Contents** tab in the *LabVIEW Help* for more information about source control in LabVIEW.

After you configure LabVIEW to use source control, you can configure source control settings on individual LabVIEW projects. Configuring source control settings on an individual LabVIEW project is useful if you want to use a different source control project than the one you specify for the LabVIEW environment, or if you do not want to use source control with a particular LabVIEW project.

LabVIEW projects use the source control configuration you specify for the LabVIEW environment by default. You cannot change the source control provider for a LabVIEW project. Source control settings for LabVIEW projects must match the provider you specify for the LabVIEW environment. You can use only one source control provider at a time.

371780C-01 Aug07