LabVIEW™ Upgrade Notes

These upgrade notes describe the process of upgrading LabVIEW for Windows, Mac OS X, and Linux to LabVIEW 2012. Before you upgrade, read this document for information about the following topics:

- The recommended process for upgrading LabVIEW
- Potential compatibility issues you should know about prior to loading any VIs you saved in a previous version of LabVIEW
- New features and behavior changes in LabVIEW 2012

Contents

Upgrading to LabVIEW 2012

Although you can upgrade small applications to a new version of LabVIEW by installing the new version and then loading your VIs, National Instruments recommends a more rigorous upgrade process to ensure that you can detect and correct upgrade difficulties as efficiently as possible.

Tip This process is especially beneficial for large LabVIEW applications that control or monitor critical operations; cannot afford extended down time; use multiple modules, toolkits, or drivers; or are saved in an unsupported version of LabVIEW. Refer to the National Instruments website at `ni.com/info` and enter the Info Code `lifecycle` for information about which versions of LabVIEW still receive mainstream support.

NATIONAL INSTRUMENTS™

Overview of the Recommended Upgrade Process

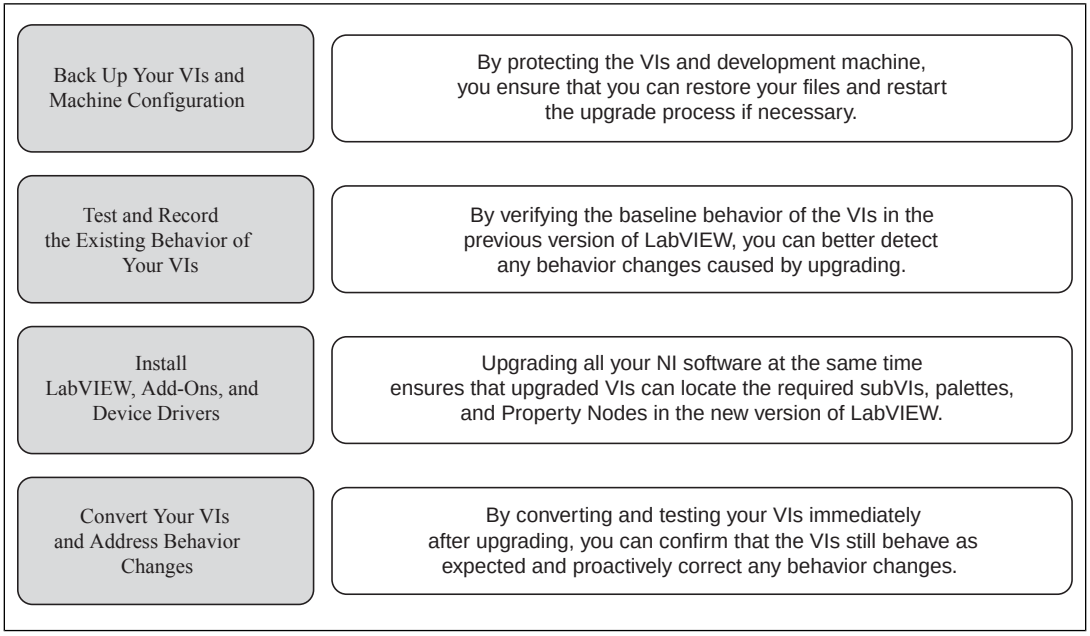

Figure 1.

Note To upgrade from LabVIEW 5.1 or earlier, you must first upgrade to an intermediate version of LabVIEW. Refer to the National Instruments website at `ni.com/info` and enter the Info Code `upgradeOld` for more information about upgrading from your specific legacy version of LabVIEW.

1. Back Up Your VIs and Machine Configuration

By protecting a copy of your VIs and, if possible, the configuration of your development or production machine before upgrading to LabVIEW 2012, you ensure that you can restore your VIs to their previous functionality and restart the upgrade process if necessary.

a. Back Up Your VIs

If you back up your VIs before you upgrade LabVIEW, you can quickly revert to the back-up copy. Without the back-up copy, you can no longer open upgraded VIs in the previous version of LabVIEW without saving each VI for the previous version.

You can back up a set of VIs using either of the following methods:

- **Submit VIs to source code control**—This action allows you to revert to this version of the VIs if you cannot address behavior changes caused by upgrading the VIs. For more information about using source code control with LabVIEW, refer to the **Fundamentals»Working with Projects and Targets»Concepts»Using Source Control in LabVIEW** topic on the **Contents** tab of the *LabVIEW Help*.
- **Create a copy of the VIs**—Create a copy of the VIs according to how they are organized:
  - Saved as a project—Open the project and select **File»Save As** to duplicate the `.lvproj` file and all project contents. Ensure that you also maintain a copy of the files on which the project depends by selecting **Include all dependencies**.

– Saved as an LLB or as VIs in a directory—From the file explorer of your operating system, create a copy of the LLB or directory and store it at a different location from the original. To prevent possible naming conflicts, avoid storing the copy on the same hard drive.

b. Back Up Your Machine Configuration

Installing a new version of LabVIEW updates shared files in ways that sometimes affect the behavior of VIs even in previous versions. However, after you update those shared files, it is very difficult to restore the previous versions of the files. Therefore, consider one of the following methods for backing up the configuration of NI software on your development machine, especially if you are upgrading from an unsupported version of LabVIEW or if down time for your applications would be costly:

- **Create a back-up image of the machine configuration**—Use *disk imaging software* to preserve the disk state of the machine before you upgrade, including installed software, user settings, and files. If you want to return the machine to its original configuration after you upgrade, deploy the back-up disk image.
- **Test the upgrade process on a test machine**—Although upgrading on a test machine requires more time than creating a back-up image, National Instruments strongly recommends this approach if you need to prevent or minimize down time for machines that control or monitor production. After resolving any issues that result from upgrading on the test machine, you can either replace the production machine with the test machine or replicate the upgrade process on the production machine.

Tip To minimize the possibility that upgraded VIs on the test machine behave differently than on the development machine, use a test machine that matches the features of the development machine as closely as possible, including CPU, RAM, operating system, and versions of software.

2. Test and Record the Existing Behavior of Your VIs

When you upgrade VIs, improvements between the previous version of LabVIEW and LabVIEW 2012 can occasionally change the behavior of the VIs. If you test the VIs in both versions, you can compare the results to detect behavior changes specifically caused by upgrading. Therefore, verify that you have current results for any of the following tests that you have available:

- Mass compile logs—Mass compiling your VIs in the previous version of LabVIEW produces a thorough log of broken VIs. This information is particularly useful if multiple people contribute to the development of the VIs or if you suspect that some of the VIs have not been compiled recently. To generate this mass compile log, place a checkmark in the **Log Results** checkbox of the **Mass Compile** dialog box. For more information about mass compiling VIs, refer to the **Fundamentals»Creating VIs and SubVIs»How-To»Saving VIs»Mass Compiling VIs** topic on the **Contents** tab of the *LabVIEW Help*.
- Unit tests that verify whether individual VIs perform their intended functions correctly
- Integration tests that verify whether a project or group of subVIs work together as expected
- Deployment tests that verify whether VIs behave as expected when deployed to a target, such as a desktop or FPGA target
- Performance tests that benchmark CPU usage, memory usage, and code execution speed. You can use the **Profile Performance and Memory** window to obtain estimates of the average execution speeds of your VIs.
- Stress tests that verify whether the VIs handle unexpected data correctly

For more information about testing VIs, refer to the **Fundamentals»Application Development and Design Guidelines»Concepts»Developing Large Applications»Phases of the Development Models»Testing Applications** topic on the **Contents** tab of the *LabVIEW Help*.

Note If you changed any VIs as the result of testing, back up the new versions of the VIs before proceeding.

3. Install LabVIEW, Add-Ons, and Device Drivers

a. Install LabVIEW, Including Modules, Toolkits, and Drivers

When you upgrade to a new version of LabVIEW, you must install not only the new development system but also modules, toolkits, and drivers that are compatible with the new version. For instructions about installing this software in the appropriate order, refer to the *LabVIEW Installation Guide*.

b. Copy user.lib Files

To ensure that custom controls and VIs you created in the previous version of LabVIEW are available to VIs in LabVIEW 2012, copy files from the `labview\user.lib` directory in the previous version to the `labview\user.lib` directory in LabVIEW 2012.

4. Convert Your VIs and Address Behavior Changes

Mass compiling your VIs in LabVIEW 2012 converts the VIs to the new version of LabVIEW and creates an error log to help you identify VIs that are broken. You can use this information in conjunction with the *Upgrade and Compatibility Issues* section of this document to identify and correct behavior changes associated with the new version of LabVIEW.

a. Mass Compile Your VIs in the New Version of LabVIEW

Mass compiling VIs simultaneously converts and saves the VIs in LabVIEW 2012. However, after mass compiling the VIs, you no longer can open the VIs in a previous version of LabVIEW without selecting **File»Save for Previous Version** for each VI or project. Therefore, mass compile only the VIs that you want to convert to the new version of LabVIEW. To help identify any problems that arose from upgrading, create a mass compile log by placing a checkmark in the **Log Results** checkbox of the **Mass Compile** dialog box.

For more information about mass compiling VIs, refer to the following topics on the **Contents** tab of the *LabVIEW Help*:

- **Fundamentals»Creating VIs and SubVIs»How-To»Saving VIs»Mass Compiling VIs**
- **Fundamentals»Creating VIs and SubVIs»How-To»Saving VIs»Common Mass Compile Status Messages**

b. Fix Any Broken VIs

Improvements between your previous version of LabVIEW and LabVIEW 2012 can occasionally cause some VIs to break if they use outdated features. To quickly identify and fix broken VIs in LabVIEW 2012, complete the following steps:

1. To identify VIs that broke during upgrading, compare the mass compile error log you created in the previous step to the log you created when testing the existing behavior of the VIs.
2. To determine whether updates to LabVIEW caused each VI to break, refer to the *Upgrade and Compatibility Issues* section of this document.

c. Identify and Correct Behavior Changes

Although National Instruments invests significant effort to avoid changing the behavior of VIs between different versions of LabVIEW, improvements and bug fixes occasionally do alter the behavior of VIs.

To quickly identify whether the new version of LabVIEW changes the behavior of your VIs, use one or more of the following tools:

- **Upgrade VI Analyzer Tests**—For large sets of VIs, these tests provide an efficient way to identify many behavior changes caused by upgrading. Complete the following steps to obtain and use these tests:
  1. Download the Upgrade VI Analyzer Tests for all versions of LabVIEW later than your previous version by referring to the National Instruments website at `ni.com/info` and entering the Info Code `analyzevi`.
  2. Open and run the tests by selecting **Tools»VI Analyzer »Analyze VIs** and starting a new VI Analyzer task. To analyze an entire project at once, select this menu option from the **Project Explorer** window rather than from a single VI.
  3. Resolve test failures by referring to the *Upgrade and Compatibility Issues* section for the version of LabVIEW that corresponds to the tests. For example, if the LabVIEW 2010 Upgrade VI Analyzer tests locate a potential behavior change, refer to the *Upgrading from LabVIEW 2009* section of that topic.
- **Upgrade documentation**
  - *Upgrade and Compatibility Issues* section of this document—Lists changes that may break or affect the behavior of your VIs. Refer to the subsections for each version of LabVIEW beginning with your previous version.

    Tip To quickly locate deprecated objects and other objects mentioned in the *Upgrade and Compatibility Issues* section, open your upgraded VIs and select **Edit»Find and Replace**.

  - LabVIEW 2012 Known Issues list—Lists bugs discovered before and throughout the release of LabVIEW 2012. Refer to the National Instruments website at `ni.com` and enter the Info Code `lv2012ki` to access this list. Refer to the *Upgrade - Behavior Change* and *Upgrade - Migration* sections to identify workarounds for any bugs that may affect the behavior of upgraded VIs.
  - Module and toolkit documentation—For some modules and toolkits, such as LabVIEW FPGA and the LabVIEW Real-Time Module, lists upgrade issues specific to that add-on
  - Driver readme files—Lists upgrade issues specific to each driver. To locate each readme, refer to the installation media for the driver.

    Tip To determine whether a behavior change resulted from a driver update rather than an update to LabVIEW, test your VIs in the previous version of LabVIEW after installing LabVIEW 2012.

- **Your own tests**—Perform the same tests on the VIs in LabVIEW 2012 that you performed in the previous version and compare the results. If you identify new behaviors, refer to the upgrade documentation to diagnose the source of the change.

Troubleshooting Common Upgrade Issues

Refer to the **Upgrading to LabVIEW 2012»Troubleshooting Common Upgrade Issues** topic on the **Contents** tab of the *LabVIEW Help* for more information about solving the following upgrade issues:

- Locating missing module or toolkit functionality
- Locating missing subVIs, palettes, and Property Nodes
- Determining why LabVIEW 2012 cannot open VIs from a previous version of LabVIEW
- Determining which versions of NI software are installed

- Restoring VIs to a previous version of LabVIEW

Upgrade and Compatibility Issues

Refer to the following sections for changes specific to different versions of LabVIEW that may break or alter the behavior of your VIs.

Refer to the `readme.html` file in the `labview` directory for information about known issues in the new version of LabVIEW, additional compatibility issues, and information about late-addition features in LabVIEW 2012.

Upgrading from LabVIEW 8.5 or Earlier

Refer to the National Instruments website at `ni.com/info` and enter the Info Code `oldUpgradeIssues` to access upgrade and compatibility issues you might encounter when you upgrade to LabVIEW 2012 from LabVIEW 8.5 or earlier. Also, refer to the other *Upgrading from LabVIEW x* sections in this document for information about other upgrade issues you might encounter.

Upgrading from LabVIEW 8.6

You might encounter the following compatibility issues when you upgrade to LabVIEW 2012 from LabVIEW 8.6. Refer to the *Upgrading from LabVIEW 2009*, *Upgrading from LabVIEW 2010*, and *Upgrading from LabVIEW 2011* sections of this document for information about other upgrade issues you might encounter.

VI and Function Behavior Changes

The behavior of the following VIs and functions changed in LabVIEW 2009 and later.

Bluetooth VIs and Functions

You must have Windows XP Service Pack 3 or later installed to use the VIs and functions on the Bluetooth palette.

Signal Generation VIs

The following VIs on the Signal Generation palette are rewritten in LabVIEW 2009 and later. To use the new functionality, replace these VIs with VIs of the same name from the Functions palette:

- Bernoulli Noise
- Binary MLS
- Binomial Noise
- Gamma Noise
- Gaussian White Noise
- Poisson Noise
- Uniform White Noise

Miscellaneous VI and Function Behavior Changes

LabVIEW 2009 and later includes the following miscellaneous VI and function behavior changes:

- If you wire a value that has a unit with an odd exponent to the square root function, the wire breaks because LabVIEW does not support units with fractional exponents.
- The Bessel Coefficients VI is rewritten to implement cutoff frequencies more accurately. As a result, the Bessel Coefficients VI and any calling VIs might run more slowly than in previous versions of LabVIEW.
- LabVIEW deploys Web services to version-specific directories. For example, a typical root location for deployed Web services in LabVIEW 2009 is `C:\Documents and Settings\All Users\Application Data\National Instruments\Web Services 2009 32-bit`. You must redeploy any Web services created in a previous version of LabVIEW to use the Web services

in LabVIEW 2009 or later. To delete a Web service deployed by a previous version of LabVIEW, you must manually remove it from the deployed location.

- The Integral x(t) VI is rewritten. To use the new functionality, replace this VI with the Integral x(t) VI from the Functions palette.
- The **section** and **refnum** inputs of the Get Key Names VI are required inputs.
- The **refnum** input of the Get Section Names VI is a required input.
- The **refnum** input of the Not A Config Data Refnum VI is a required input.
- You no longer can use the Check if File or Folder Exists VI to check for paths in stand-alone applications or shared libraries.

Deprecated VIs and Functions

LabVIEW 2009 and later do not support the following VIs and functions:

- **LToCStr**—Use the LToCStrN function instead. The LToCStrN function differs from the LToCStr function because it takes a parameter specifying the size of the C string buffer to which LabVIEW copies the string. These functions are LabVIEW Manager functions. The Code Interface Node is deprecated in LabVIEW 2010 and later. Use the Call Library Function Node instead.
- **Open Config Data (compatibility)**—Use the Open Config Data VI instead. The Open Config Data VI differs from the Open Config Data (compatibility) VI because the Open Config Data VI includes the **file created?** output.
- **Sound VIs (Mac OS X)**—Use the Sound VIs instead. LabVIEW 2009 and later support the same API for Windows, Mac OS X, and Linux.
- **Unconstrained Exponential Fit**—Use the Exponential Fit VI instead. The Exponential Fit VI differs from the Unconstrained Exponential Fit VI because the Exponential Fit VI does not include the **refine?** input but does include the **parameter bounds** input and **offset** output.
- **Unconstrained Gaussian Peak Fit**—Use the Gaussian Peak Fit VI instead. The Gaussian Peak Fit VI differs from the Unconstrained Gaussian Peak Fit VI because the Gaussian Peak Fit VI includes the **parameter bounds** input and **offset** output.

Deprecated Properties, Methods, and Events

LabVIEW 2009 and later do not support the following properties, methods, and events:

- Bus Name property of the DigitalGraph class. Use the Plot Name property instead.
- Callees' Names property of the VI class. Use the Get VI Dependencies (Names and Paths) method instead. The Get VI Dependencies (Names and Paths) method provides the same functionality as the Callees' Names property when you use the default values for all input parameters.
- Callees property of the VI (ActiveX) class.

Renamed Properties, Methods, and Events

- In LabVIEW 2009 and later, the XML Parser classes do not include XML in their names. For example, XML_Attributes becomes Attributes.
- The following properties, methods, and events are renamed in LabVIEW 2009 and later.

Class	LabVIEW 8.6 Name	LabVIEW 2009 and Later Name	Type
Document	Do Namespaces	Process Namespaces	Property
Document	Do Schema	Process Schema	Property
Variable	Alarming:BadStatus:AckType	Alarming:BadStatus:Ack Type	Property
Variable	Alarming:BadStatus:AllowLog	Alarming:BadStatus:Allow Log	Property

Class	LabVIEW 8.6 Name	LabVIEW 2009 and Later Name	Type
Variable	Alarming:Boolean:AckType	Alarming:Boolean:Ack Type	Property
Variable	Alarming:Boolean:AlarmOn	Alarming:Boolean:Alarm On	Property
Variable	Alarming:Boolean:AllowLog	Alarming:Boolean:Allow Log	Property
Variable	Alarming:Hi:AckType	Alarming:Hi:Ack Type	Property
Variable	Alarming:Hi:AllowLog	Alarming:Hi:Allow Log	Property
Variable	Alarming:HiHi:AckType	Alarming:HiHi:Ack Type	Property
Variable	Alarming:HiHi:AllowLog	Alarming:HiHi:Allow Log	Property
Variable	Alarming:Lo:AckType	Alarming:Lo:Ack Type	Property
Variable	Alarming:Lo:AllowLog	Alarming:Lo:Allow Log	Property
Variable	Alarming:LoLo:AckType	Alarming:LoLo:Ack Type	Property
Variable	Alarming:LoLo:AllowLog	Alarming:LoLo:Allow Log	Property
Variable	Alarming:RateOfChange:AckType	Alarming:RateOfChange:Ack Type	Property
Variable	Alarming:RateOfChange:AllowLog	Alarming:RateOfChange:Allow Log	Property
Variable	Alarming:U32BitField:AckType	Alarming:U32BitField:Ack Type	Property
Variable	Alarming:U32BitField:AlarmOn	Alarming:U32BitField:Alarm On	Property
Variable	Alarming:U32BitField:AllowLog	Alarming:U32BitField:Allow Log	Property
Variable	Alarming:U32BitField:SelectMask	Alarming:U32BitField:Select Mask	Property
Variable	Logging:LogData	Logging:Log Data	Property
Variable	Logging:LogEvents	Logging:Log Events	Property
Variable	Logging:TimeResolution	Logging:Time Resolution	Property
Variable	Logging:ValueResolution	Logging:Value Resolution	Property
Variable	Network:AccessType	Network:Access Type	Property
Variable	Network:BuffSize	Network:Buffer Size	Property
Variable	Network:ElemSize	Network:Element Size	Property
Variable	Network:PointsPerWaveform	Network:Points Per Waveform	Property
Variable	Network:ProjectBinding	Network:Project Binding	Property
Variable	Network:ProjectPath	Network:Project Path	Property
Variable	Network:UseBinding	Network:Use Binding	Property
Variable	Network:UseBuffering	Network:Use Buffering	Property
Variable	Real-Time:ArrayLength	Real-Time:Array Length	Property
Variable	Real-Time:BufferLength	Real-Time:Buffer Length	Property

Class	LabVIEW 8.6 Name	LabVIEW 2009 and Later Name	Type
Variable	Real-Time:DatapointsInWaveform	Real-Time:Datapoints In Waveform	Property
Variable	Real-Time:UseBuffering	Real-Time:Use Buffering	Property
Variable	Scaling:EngineeringMax	Scaling:Engineering Max	Property
Variable	Scaling:EngineeringMin	Scaling:Engineering Min	Property
Variable	Scaling:InvertMask	Scaling:Invert Mask	Property
Variable	Scaling:RawMax	Scaling:Raw Max	Property
Variable	Scaling:RawMin	Scaling:Raw Min	Property
Variable	Scaling:SelectMask	Scaling:Select Mask	Property

Application Builder Changes

LabVIEW 2009 and later include the following Application Builder changes.

File Layout Changes

In LabVIEW 8.6, the Application Builder saves VIs and library files in a flat list within the application and saves VIs with conflicting filenames outside the application in separate directories. In LabVIEW 2009 and later, the Application Builder stores source files within the application using a layout similar to the directory structure of the source files on disk. This internal file layout preserves source file hierarchy inside the application.

If you call VIs dynamically, use relative paths to ensure the application loads the VIs correctly at run time.

Custom Configuration File Changes

In LabVIEW 8.6 and earlier, when you build a stand-alone application that includes a custom configuration file, LabVIEW appends LabVIEW environment settings to the existing contents of the file when the following conditions are true:

- The custom configuration file has the same name as the application.
- The custom configuration file is in the same directory as the application.
- In the build specification for the application, the **Use custom configuration file** checkbox on the **Advanced** page of the **Application Properties** dialog box does not contain a checkmark.

When these conditions are true in LabVIEW 2009 and later, LabVIEW overwrites the contents of custom configuration files with LabVIEW environment settings.

Case Structure Output Tunnel Changes

LabVIEW 2009 and later determine the data type from a Case structure output tunnel by using a data type that can handle all cases in the structure, including cases that never execute. For example, consider a Case structure with two cases, TRUE and FALSE. In the TRUE case, an 8-bit unsigned integer is wired to an output tunnel. In the FALSE case, a 32-bit unsigned integer is wired to the output tunnel. In LabVIEW 8.5.*x* and 8.6.*x*, if you wire a constant to select the TRUE case, the data type from the output tunnel is 8-bit unsigned integer because the constant prevents the FALSE case from executing. In LabVIEW 2009 and later, if you wire a constant to select the TRUE case, the data type from the output tunnel is 32-bit unsigned integer.

This change in behavior might cause VIs created in LabVIEW 8.5.*x* and 8.6.*x* to break in LabVIEW 2009 and later if the output data type is a fixed-point number or fixed-sized array.

Custom Icon Editor VI Changes

In LabVIEW 8.6 and earlier, when LabVIEW calls a VI that is a custom icon editor, LabVIEW automatically opens the front panel of the VI. In LabVIEW 2009 and later, you must configure a VI that is a custom icon editor to open its own front panel on call. For simple VIs that do not need to rearrange their front panels before they open, use the Execution:Show Front Panel on Call property. For more complex VIs that need to rearrange their front panels before they open, use the Front Panel:Open method.

Custom Probes Changes (Linux)

Custom probes you save in LabVIEW 8.6 and earlier do not open in LabVIEW 2009 and later. You must manually copy the custom probes from the `LabVIEW Data` directory of the previous version of LabVIEW into the `LabVIEW Data` directory of LabVIEW 2009 and later. You can find the `LabVIEW Data` directory for LabVIEW 2009 and later at `/home/<username>/LabVIEW Data`.

.NET Changes

Creating and communicating with .NET objects requires the .NET Framework 2.0 or later.

LabVIEW MathScript Changes

LabVIEW MathScript is no longer a part of the Full and Professional Development Systems. In LabVIEW 2009 and later, LabVIEW MathScript becomes the LabVIEW MathScript RT Module. You cannot run VIs from previous versions of LabVIEW that contain MathScript Nodes until you install and activate the MathScript RT Module or remove the MathScript Nodes from the VIs. If you have already purchased the MathScript RT Module, select **Help»Activate LabVIEW Components** to activate the product.

Upgrading from LabVIEW 2009

You might encounter the following compatibility issues when you upgrade to LabVIEW 2012 from LabVIEW 2009. Refer to the *Upgrading from LabVIEW 2010* and *Upgrading from LabVIEW 2011* sections of this document for information about other upgrade issues you might encounter.

VI and Function Behavior Changes

The following VIs use a higher attenuation than the value of the **stopband attenuation** input to design an elliptic filter when the filter **order** is high:

- Elliptic Coefficients
- Elliptic Filter
- Elliptic Filter PtByPt

VISA Find Resource Function

In LabVIEW 2010 and later, the VISA Find Resource function returns error code –1073807343 if the system does not locate any devices.

Deprecated VIs, Functions, and Nodes

LabVIEW 2010 and later do not support the following VIs, functions, and nodes:

- **Code Interface Node**—Use the Call Library Function Node instead.
- **Convert TDM to TDMS**—Use the Convert to TDM or TDMS VI instead. This VI converts a file to the `.tdm` or `.tdms` file format.
- **Convert TDMS to TDM**—Use the Convert to TDM or TDMS VI instead.
- **Get Property Type**—Use the Get Property Info VI instead. This VI returns information about the properties of a data file, channel group, or channel.

- **FFT Power Spectrum**—Use the FFT Power Spectrum and PSD VI instead.
- **FFT Power Spectral Density**—Use the FFT Power Spectrum and PSD VI instead.
- **List Properties**—Use the Get Property Info VI instead.
- **Merge Errors** VI—Use the **Merge Errors** function instead.
- **Merge Queries**—Use the Merge Storage Refnums VI instead.

Floating-Point Math Operations

Due to changes to the LabVIEW compiler, the results of several mathematical operations performed using floating-point numbers might differ from results returned in previous versions of LabVIEW. The accuracy of algorithms written in LabVIEW using floating-point numbers is the same and in many cases improved in LabVIEW 2010 and later. However, in a few operations the results might be less accurate than in previous versions. LabVIEW 2010 and later implement functions internally with the same numeric precision as the input data types, whereas previous versions implemented functions with a higher numeric precision than the input data types. The acceptable error for the results of these operations is still appropriate for the data types of the inputs.

Note Refer to the National Instruments website at `ni.com/info` and enter the Info Code `exdj8b` for more information about mathematical operations using floating-point numbers.

Creating LabVIEW Classes

In LabVIEW 2009 and earlier, you can create a class with a strictly typed VI refnum that includes itself or a child class in the connector pane of the VI. In LabVIEW 2010 and later, the class breaks unless you use a VI refnum that is not strictly typed or remove the VI refnum from the private data control.

Building an Installer (Windows)

In LabVIEW 2010 and later, if you load a project with an installer that requires Windows 2000 or later, LabVIEW updates the system requirements to Windows XP or later. After you install LabVIEW 2010 and later, you cannot use a previous version of LabVIEW on the computer to build installers that run on Windows 2000.

Using the Correct Calling Convention in a Call Library Function Node

In LabVIEW 8.5, LabVIEW 8.6, and LabVIEW 2009, when you specify the incorrect calling convention for a Call Library Function Node, LabVIEW recovers from the error and uses the correct calling convention. LabVIEW 2010 and LabVIEW 2011 do not perform this check, which requires you to select the correct calling convention yourself. Therefore, if you convert VIs that contain Call Library Function Nodes from LabVIEW 8.5, LabVIEW 8.6, or LabVIEW 2009 to LabVIEW 2010 or LabVIEW 2011, the VIs will crash if they have the incorrect calling convention selected.

Complete the following steps to prepare a VI that contains Call Library Function Nodes to be converted to LabVIEW 2010 or LabVIEW 2011:

1. Open the VI in the LabVIEW version in which it was last saved.
2. Right-click each Call Library Function Node and select **Configure** from the shortcut menu to display the **Call Library Function** dialog box.
3. Click the **Error Checking** tab.
4. Place a checkmark in the **Maximum** checkbox to enable maximum error checking. This selection causes LabVIEW to notify you at run time if you select the incorrect calling convention.
5. Click the **OK** button.
6. After you select maximum error checking for each Call Library Function Node, run the VI.
7. Select the correct calling convention for each Call Library Function Node that returns an error.

After you resolve all calling convention errors, you can convert the VI to LabVIEW 2010 or LabVIEW 2011.

Upgrading from LabVIEW 2010

You might encounter the following compatibility issues when you upgrade to LabVIEW 2012 from LabVIEW 2010. Refer to the *Upgrading from LabVIEW 2011* section of this document for information about other upgrade issues you might encounter.

VI and Function Behavior Changes

In LabVIEW 2011 and later, the **multicast addr** input of the UDP Multicast Open VI is a required input. Also, the **port** output is renamed **port out**.

Deprecated VIs, Functions, and Nodes

In LabVIEW 2011 and later, the Zero Phase Filter VI no longer has the **init/cont** input in any of its polymorphic instances. To use the new version of the VI, replace instances of the Zero Phase Filter VI from previous versions of LabVIEW with the VI of the same name from the Filters palette.

Property, Method, and Event Behavior Changes

The behavior of the following properties, methods, and events changed in LabVIEW 2011 and later:

- In LabVIEW 2010, the Clear Compiled Object Cache method clears the object cache associated with a specific target. In LabVIEW 2011 and later, the Clear Compiled Object Cache method clears the entire user cache for the running version of LabVIEW. Therefore, although VIs created in LabVIEW 2010 that contain the Clear Compiled Object Cache method do not break in LabVIEW 2011 and later, they delete more VI object files than they did previously, which causes the associated VIs to recompile when loaded.
- In LabVIEW 2010 and earlier, the **NewRange** event data field for the Scale Range Change event ignores custom offset and multiplier values you set for a graph or chart. In LabVIEW 2011 and later, the **NewRange** event data field factors in custom offset and multiplier values in the results it returns. If you use code to work around this issue in LabVIEW 2010 or earlier, you must update the upgraded version of the code.

Deprecated Properties, Methods, and Events

LabVIEW 2011 and later do not support the Subsystem From Selection method of the SimDiagram class.

Migrating Build Specifications for Targets That Do Not Support SSE2 Instructions

To migrate a build specification for a target that does not support SSE2 instructions to LabVIEW 2011 or later, you must disable the SSE2 optimizations for the build specification. If you do not disable the optimizations, LabVIEW still allows you to build the associated application, but the application cannot run on the intended target.

Refer to the **Fundamentals»Building and Distributing Applications»Configuring Build Specifications»Verifying That Target Hardware Supports SSE2 Instructions** topic on the **Contents** tab of the *LabVIEW Help* for information about which hardware types support SSE2 instructions.

Polymorphic VI Terminals That Support 16-bit, 64-bit and Double-Precision Numeric Data Types

LabVIEW 2011 and later coerce certain numeric data to double-precision numeric data in the following instances:

- If you wire extended-precision numeric data to the terminal of a polymorphic VI that supports both the double-precision numeric and 64-bit integer data types, LabVIEW coerces the extended-precision numeric data to double-precision numeric data.

- If you wire 64-bit integer numeric data to the terminal of a polymorphic VI that supports both the double-precision numeric and 16-bit integer data types, LabVIEW coerces the 64-bit integer data to double-precision data.

This behavior matches the behavior in LabVIEW 8.5 and 8.6. However, in LabVIEW 8.2, 2009, and 2010, LabVIEW selects the 64-bit integer or 16-bit integer data type instead of the double-precision data type.

Improved Error Reporting for Certain LabVIEW Shared Libraries

When you call a LabVIEW shared library with the Call Library Function Node in previous versions of LabVIEW, the shared library fails to execute on target computers that do not have required resources installed. However, in those situations, the shared libraries do not automatically return an error or otherwise indicate that execution failed. In LabVIEW 2011 and later, when the Call Library Function Node calls these shared libraries, LabVIEW returns an error to indicate the failure. Therefore, affected LabVIEW shared libraries that misleadingly do not return an error in LabVIEW 2010 *do* return an error in LabVIEW 2011 and later.

This error-reporting enhancement affects, but is not limited to, VIs that call LabVIEW shared libraries with any of the following characteristics:

- A VI inside the shared library uses licensed features that are not installed on the target computer.
- A VI inside the shared library uses a Call Library Function Node whose associated shared library is not installed on the target computer.
- The VIs inside the shared library were compiled using the SSE2 optimizations but the target computer does not support SSE2 instructions.

Changes to the Locations LabVIEW Searches for NI Example Finder Data Files

LabVIEW 2011 and later search for NI Example Finder data files (`.bin3`) in fewer locations than previous versions of LabVIEW. To ensure LabVIEW finds example VIs you create for the NI Example Finder, you must place the `.bin3` files in one of the following directories:

- `labview\examples\exbins`—Previous versions of LabVIEW allowed you to place the `.bin3` files anywhere within the `examples` directory.
- `labview\instr.lib`
- `labview\user.lib`

Upgrading from LabVIEW 2011

You might encounter the following compatibility issues when you upgrade to LabVIEW 2012 from LabVIEW 2011.

Deprecated VIs, Functions, and Nodes

LabVIEW 2012 does not support the following VIs, functions, and nodes:

- **Polar Plot**—Use the Polar Plot with Point Options VI instead. The Polar Plot with Point Options VI provides two new inputs, **Lines/Points** and **Size**.
- **Draw Rect**—Use the Draw Rectangle VI instead.

Property, Method, and Event Behavior Changes

In the Set Cell Value method of the Table class, the **X Index** and **Y Index** inputs changed from 32-bit unsigned integers to 32-bit signed integers.

Deprecated Properties, Methods, and Events

LabVIEW 2012 does not support the following properties, methods, and events:

- Create from Data Type method of the Diagram class. If you upgrade a VI that contains this method, the VI now calls the Create from Data Type (Deprecated) method. Replace the deprecated method with the new Create from Data Type method, which no longer includes the **style** input.
- Frames[] property of the TimeFlatSequence class. Use the Frames[] property of the FlatSequence class instead.
- Front Panel Window:Open property of the VI class. Use the Front Panel:Open method, the Front Panel:Close method, or the Front Panel Window:State property instead.
- FPWinOpen property of the VI (ActiveX) class. Use the OpenFrontPanel method, the CloseFrontPanel method, or the FPState property instead.
- Static Member VIs property of the LVClassLibrary class. Use the new version of the Static Member VIs property instead.
- Dynamic Member VIs property of the LVClassLibrary class. Use the new version of the Dynamic Member VIs property instead.

Renamed Properties, Methods, and Events

The following properties, methods, and events are renamed in LabVIEW 2012.

Class	LabVIEW 2011 Name	LabVIEW 2012 Name	Type
ProjectItem	Children[]	Owned Items[]	Property
ProjectItem	Parent	Owner	Property
LVClassLibrary	AncestorControlRefs	Ancestor Restricts Reference Creation	Property

LabVIEW 2012 Features and Changes

The Idea Exchange icon denotes a new feature idea that originates from a product feedback suggestion on the LabVIEW Idea Exchange discussion forums. Refer to the National Instruments website at `ni.com/info` and enter the Info Code `ex3gus` to access the NI Idea Exchange discussion forums.

Refer to the `readme.html` file in the `labview` directory for known issues, a partial list of bugs fixed, additional compatibility issues, and information about late-addition features in LabVIEW 2012.

New Example VIs

Refer to the **New Examples for LabVIEW 2012** folder on the **Browse** tab of the NI Example Finder to view descriptions for and launch example VIs added to the current version of LabVIEW.

Block Diagram Enhancements

Event-Driven Programming in the LabVIEW Base Package

The LabVIEW Base Package allows you to create and modify Event structures, events, and Events functions that detect and handle events triggered by the user.

Refer to the **Fundamentals»Event-Driven Programming** book on the **Contents** tab of the *LabVIEW Help* for more information about handling events.

[*Idea submitted by NI Discussion Forums member Mark_Yedinak.*]

Automatically Concatenating Arrays Leaving Loops

To concatenate arrays across loop iterations in previous versions of LabVIEW, you must use shift registers and the Build Array function. In LabVIEW 2012, you can automatically concatenate arrays by right-clicking the output tunnel and selecting **Tunnel Mode»Concatenating** from the shortcut menu. Selecting **Concatenating** appends all inputs in order, forming an output array of the same dimension as the array wired to the tunnel input. Additional tunnel modes include **Last Value** and **Indexing**. **Last Value** displays the value from the last loop iteration. **Indexing** builds an array of higher dimension and is the default behavior in previous versions of LabVIEW. Refer to the Find Out of Range Elements VI in the `labview\examples\general\looptunnels` directory for an example of automatically concatenating arrays. [*Idea submitted by NI Discussion Forums member tst.*]

Conditionally Processing Loop Outputs

In LabVIEW 2012, you can configure each output tunnel of a loop to omit values that meet a condition you specify. To enable this feature, right-click the loop output tunnel and select **Tunnel Mode»Conditional** from the shortcut menu.

For example, the following block diagram illustrates how the conditional output tunnel accomplishes the same goal as the Case structure in previous versions of LabVIEW.

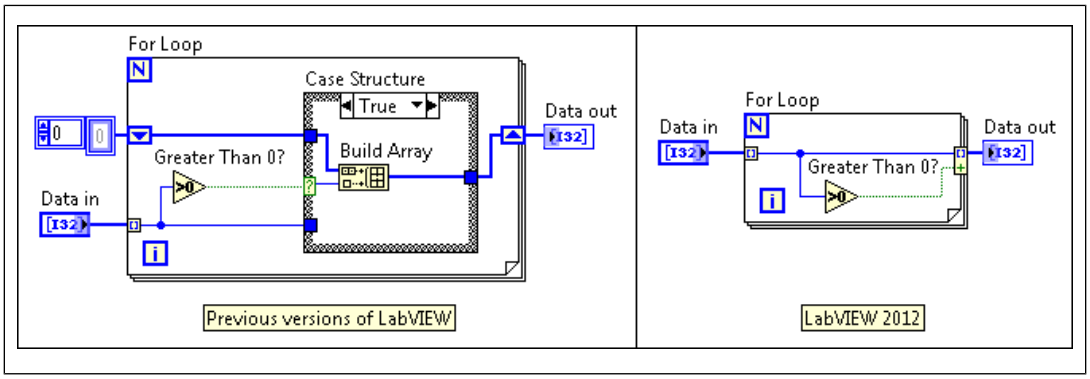

Figure 2.

You can apply this conditional terminal to all three types of output tunnels: **Last Value**, **Indexing**, and **Concatenating**.

Refer to the Find Out of Range Elements VI in the `labview\examples\general\looptunnels` directory for an example of conditionally writing values to loop output tunnels.

[*Idea submitted by NI Discussion Forums member falkpl.*]

Note The **Conditional** tunnel option performs memory allocations as often as the Build Array implementation. Therefore, just like with the Build Array function, National Instruments recommends you consider alternatives to the conditional tunnel in portions of your application where performance is critical.

Aligning Selected and Nested Labels Using Keyboard Shortcuts

While the **Quick Drop** dialog box is active, you can use keyboard shortcuts to align the labels of objects on the block diagram in the following ways:

- Select multiple objects and press <Ctrl-T>—Moves the label of each selected indicator terminal to the right of the terminal and moves the label of every other selected object to the left of the object.

- Press <Ctrl-Shift-T>—Moves the label for each control terminal to the left of the terminal and moves the label for each indicator terminal to the right of the terminal, regardless of whether the terminal is nested within the diagram of a loop or other structure.

Miscellaneous Block Diagram Enhancements

- LabVIEW 2012 allows you to remove broken wires not only from the entire block diagram but also from any area or structure you select. [*Idea submitted by NI Discussion Forums member manu.NET*.]
- To create, edit, and view long string constants without using excessive space on the block diagram, right-click the string constant and select **Edit** from the shortcut menu. This option displays the **Edit String Constant** dialog box, in which you can view and modify the entire string. [*Idea submitted by NI Discussion Forums member Broken Arrow*.]
- To more conveniently document code within structures, write comments in the new subdiagram labels that automatically move and resize with the structure. To display subdiagram labels, right-click the structure and select **Visible Items»Subdiagram Label** from the shortcut menu. You also can specify the default visibility and justification of these labels on the **Block Diagram** page of the **Options** dialog box. [*Idea submitted by NI Discussion Forums member Hueter*.]
- You can place the following new probes on the associated wire types to display the data on the wire and to allow conditional breakpoints:
  - Conditional Signed8 Probe
  - Conditional Signed16 Probe
  - Conditional Unsigned8 Probe
  - Conditional Unsigned16 Probe

Environment Enhancements

Accessing Resources and Creating Projects from the Getting Started Window

The redesigned **Getting Started** window allows you to more easily access the resources you need to use LabVIEW. The new design emphasizes common tasks, such as creating projects and opening existing files, and presents less common tasks, such as downloading additional drivers and products, through submenus.

Identifying Long Paths in Controls, Indicators, and Constants

When a path is too long for a control, indicator, or constant to display, LabVIEW 2012 replaces the middle of the path with an ellipsis (. . .) to notify the user that part of the path is hidden. To configure LabVIEW to display long paths differently, use the **Scrollbar Visibility** field on the **Appearance** page of the **Properties** dialog box. [*Idea submitted by NI Discussion Forums member blawson.]*

Viewing Data Types of Parameters in the Context Help Window

To provide quick access to the data type associated with each terminal of a VI or function, the **Context Help** window contains the following new fields when you hover over a terminal:

- **Terminal Data Type**—Displays the default data type the terminal accepts.
- **Connected Wire Data Type**—When the terminal displays a coercion dot, displays the data type of the wire connected to the terminal rather than the data type expected by the terminal.

[Idea submitted by NI Discussion Forums member Dany Allard.]

Resolving Delays When Saving or Loading Large VIs

To resolve delays that occur when LabVIEW compiles a large or complex VI, you can configure LabVIEW 2012 to devote more resources to editor responsiveness. However, reallocating these resources prevents LabVIEW from fully optimizing the VI for execution speed. Therefore, to both reduce delays

and preserve the execution speed of the VI, National Instruments recommends that you try dividing the VI into subVIs before changing how LabVIEW prioritizes editor responsiveness.

Refer to the **Fundamentals»Managing Performance and Memory»How-To»Choosing between Editor Responsiveness and VI Execution Speed** topic on the **Contents** tab of the *LabVIEW Help* for more information about when and how to configure LabVIEW to prioritize editor responsiveness differently.

Separating Compiled Code from Additional File Types

LabVIEW 2012 adds the ability to separate compiled code from the following types of files:

- VIs that contain Express VIs
- Custom controls and global variables
- LabVIEW classes and their private data controls
- Project libraries and statechart libraries
- XControls

Separate compiled code from VIs and other file types to obtain various source control benefits and possible load time improvements for entire projects and VI hierarchies.

Refer to the **Fundamentals»Working with Projects and Targets»Concepts»Using Source Control in LabVIEW»Facilitating Source Control by Separating Compiled Code from VIs and Other File Types** topic on the **Contents** tab of the *LabVIEW Help* for more information about the benefits of and procedures for separating compiled code from LabVIEW files.

Dialog Box Enhancements

- The **Edit Items** page of the **Properties** dialog box for ring and enumerated type controls includes the following improvements to the user interface:
  - To select multiple items, you can press and hold the <Ctrl> key while selecting the items. **(Mac OS X)** Press the <Command> key. **(Linux)** Press the <Alt> key.
  - To rearrange items, you can drag and drop the items rather than clicking the **Move Up** and **Move Down** buttons.
  - To delete items, you can press the <Delete> key rather than clicking the **Delete** button.
  - To insert items, you can press the <Insert> key rather than clicking the **Insert** button.
  - To prevent items from appearing in the control, you can click the **Disable Items** button.

  [*Idea submitted by NI Discussion Forums member Intaris*.]
- Panes have a redesigned **Properties** dialog box that separates the background settings into a new **Background** page.
- LabVIEW 2012 reorganizes the reentrancy options on the **Execution** page of the **VI Properties** dialog box as a set of three radio buttons in which the default setting is always **Non-reentrant execution**. If you want to enable reentrant execution, this design encourages you to consider the tradeoffs and ramifications of each reentrancy option rather than relying on LabVIEW to make a default choice that may not be optimal for your application. Refer to the **Fundamentals»Managing Performance and Memory»Concepts»Reentrancy: Allowing Simultaneous Calls to the Same SubVI** topic on the **Contents** tab of the *LabVIEW Help* for more information about the effects of each reentrancy option on your application.

Miscellaneous Environment Enhancements

- You can perform the same operation on several objects simultaneously by selecting the objects, right-clicking any one of them, and choosing an operation from the shortcut menu. For example, you can use this technique to display labels or toggle the **View As Icon** setting for multiple block

diagram terminals at the same time. [*Ideas submitted by NI Discussion Forums members JackDunaway and muks.*]

- Use the Error Ring to quickly select and pass NI or custom error codes throughout your VI. When you click the pull-down menu in the Error Ring, you can select the error code from a dialog box rather than entering it manually.
- You can specify a different default label position for the following sets of objects: controls and constants, indicators, and all other objects. To adjust these default label positions, use the **Default label position** pull-down menus on the **Front Panel** and **Block Diagram** pages of the **Options** dialog box. [*Idea submitted by NI Discussion Forums member Broken Arrow.*]

Application Builder Enhancements

Improvements to .NET Interop Assemblies Generated by LabVIEW

In LabVIEW 2012, new build specifications for .NET interop assemblies produce assemblies with the following improvements:

- Refnums exposed in method signatures are type safe. The assembly includes a class definition for each exposed refnum type.
- Inputs appear before outputs in method signatures.
- If a VI belongs to a project library, the .NET method generated from that VI belongs to a class with the same name as the project library. For nested project libraries, LabVIEW generates nested classes. If a VI does not belong to a project library, the .NET method generated from that VI belongs to the class specified in the **.NET interop assembly class name** section on the **Information** page of the **.NET Interop Assembly Properties** dialog box.
- Method signatures no longer include any **error in** and **error out** parameters associated with the corresponding VI.
- The VI description appears as IntelliSense documentation in Microsoft Visual Studio for the corresponding method signature. You also can add custom documentation for a method in the **Define VI Prototype** dialog box.

To generate .NET interop assemblies without the LabVIEW 2012 improvements, place a checkmark in the **LabVIEW 2011 compatibility mode** checkbox on the **Advanced** page of the **.NET Interop Assembly Properties** dialog box.

If you migrate a build specification created in an earlier version of LabVIEW, the **LabVIEW 2011 compatibility mode** checkbox contains a checkmark by default to ensure that the generated assembly is still compatible with pre-existing external code. If you disable the **LabVIEW 2011 compatibility mode** checkbox for these migrated build specifications, you must modify external code to properly interact with the updated assembly that is generated by the build specification.

Improvements to Shared Libraries Generated by LabVIEW

In LabVIEW 2012, new build specifications for shared libraries produce shared libraries with the following improvements:

- LabVIEW generates qualified C function names for the VIs of a project. A qualified name is a name based on the owning library hierarchy of a VI in addition to the VI name. For example, if `Foo.vi` belongs to `FooLib.lvlib`, LabVIEW generates the C function name as `FooLib_Foo`.
- Instead of TD1, TD2, and so on, LabVIEW generates more meaningful type names for enums, clusters, and arrays exposed as handles depending on how these parameters are defined in the exported VI:
  - Defined as a type definition—LabVIEW uses the name of the type definition as the type name in the generated header file, qualifying it with the name of the containing library.

  - Not defined as a type definition—LabVIEW uses an appropriate name, such as Enum, Cluster, or ClusterArray, as the type name in the generated header file.
- LabVIEW exposes LabVIEW enum values in a generated header file.
- Function prototypes no longer include **error in** and **error out** parameters. Generated functions return the error code of the first **error out** parameter as the return value.
- To manage memory allocation for LabVIEW array types exposed as handles, the LabVIEW-generated shared library provides `Allocate`, `DeAllocate`, and `Resize` functions for each array type exposed as a handle. Open the generated header file to view the function prototypes specific to the generated shared library.
- The VI description appears as C function documentation above the function prototype in the generated header file. You also can add custom documentation for a function in the **Define VI Prototype** dialog box.

To generate shared libraries without the LabVIEW 2012 improvements, place a checkmark in the **LabVIEW 2011 compatibility mode** checkbox on the **Advanced** page of the **Shared Library Properties** dialog box.

If you migrate a build specification created in an earlier version of LabVIEW, the **LabVIEW 2011 compatibility mode** checkbox contains a checkmark by default to ensure that the generated shared library is still compatible with pre-existing external code. If you disable the **LabVIEW 2011 compatibility mode** checkbox for these migrated build specifications, you must modify external code to properly interact with the updated shared library that is generated by the build specification.

Improvements to Installers Generated by LabVIEW

When you build an installer, you can include an executable to run when the user begins uninstallation. Use the **Advanced** page of the **Installer Properties** dialog box to determine which executable to run.

Performance Improvements to Built Applications

To improve the load time and reduce memory usage of stand-alone applications, .NET interop assemblies, shared libraries, source distributions, and Web services that include inline subVIs, place a checkmark in the **Disconnect unused inline subVIs** checkbox on the **Additional Exclusions** page of the build specification. This checkbox prevents LabVIEW from loading unused inline subVIs into memory when you load the resulting built application. LabVIEW considers an inline subVI to be unused if VIs within the build only call the subVI statically via the subVI node. On the other hand, if a VI in the build refers to the subVI via a Static VI Reference function, LabVIEW cannot disconnect the subVI.

Refer to the **Fundamentals»Managing Performance and Memory»Concepts»VI Execution Speed** topic on the **Contents** tab of the *LabVIEW Help* for more information about inline subVIs.

New and Changed VIs, Functions, and Nodes

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

Computational Geometry VIs

The Computational Geometry palette includes the following new VIs:

- Convex Polygon Intersection
- Polygon Centroid

TDMS Advanced Data Reference I/O Functions

Use the TDMS Advanced Data Reference I/O functions to interact with data that is owned by a component external to LabVIEW, such as the direct memory access (DMA) buffer of a device driver that controls a data-streaming device. You can use these functions to asynchronously write data from the DMA buffer

of a device driver to `.tdms` files without copying the data into a LabVIEW array first. You also can use these functions to asynchronously read data from `.tdms` files and place the data directly into the DMA buffer.

The TDMS Advanced Data Reference I/O palette includes the following new functions:

- TDMS Advanced Asynchronous Read (Data Ref)
- TDMS Advanced Asynchronous Write (Data Ref)
- TDMS Configure Asynchronous Reads (Data Ref)
- TDMS Configure Asynchronous Writes (Data Ref)
- TDMS Get Asynchronous Read Status (Data Ref)
- TDMS Get Asynchronous Write Status (Data Ref)

Actor Framework VIs

Use the Actor Framework VIs to build applications that consist of independently-running objects, called actors, that communicate with one another. For information about the Actor Framework and examples of using these VIs, refer to the Actor Framework template and Feedback Evaporative Cooler sample project that are available in the **Create Project** dialog box.

LabVIEW 2012 also includes the **Actor Framework Message Maker** dialog box, which you use to construct messages that actors send to one another.

HTTP Client VIs

The HTTP Client palette includes the following new palettes:

- Headers
- Security

The Headers palette includes the following new VIs:

- GetHeader
- HeaderExists
- ListHeaders
- RemoveHeader

The Security palette includes the following new VIs:

- ConfigKey
- Decrypt
- Encrypt

Web Services VIs

The Web Services palette is reorganized to include the following new palettes:

- Output
- Security
- Sessions

The Security palette includes the following new VIs:

- Decrypt
- Encrypt

- Get Auth Details

FTP VIs

The Protocols palette includes the new FTP palette and VIs. In previous versions of LabVIEW, the FTP palette is part of the Internet Toolkit.

Miscellaneous New VIs and Functions

- The Advanced File palette includes the Preallocated Read from Binary File function.
- The Dialog & User Interface palette includes the Error Ring.

TDM Streaming VIs and Functions

In LabVIEW 2011 and earlier, the TDM Streaming VIs and functions support only Windows and the real-time operating systems of NI ETS and Wind River VxWorks. In LabVIEW 2012, the TDM Streaming VIs and functions support more operating systems, such as Mac OS X and Linux.

Note Not all TDM Streaming VIs and functions provide cross-platform support. Refer to the help topic for each TDM Streaming VI or function in the *LabVIEW Help* for information about the operating system that each VI or function supports.

LabVIEW categorizes the VIs and functions on the Advanced TDMS palette into the following subpalettes:

- TDMS Advanced Asynchronous I/O
- TDMS Advanced Data Reference I/O
- TDMS Advanced Synchronous I/O

The **data type** input of the TDMS Configure Asynchronous Writes function changed from required to recommended.

Miscellaneous VI and Function Changes

LabVIEW 2012 includes the following miscellaneous VI and function changes:

- **Compound Arithmetic**—If you replace an Add, Multiply, And, Or, or Exclusive Or function with a Compound Arithmetic function, the Compound Arithmetic function defaults to the operation you replaced instead of defaulting to the add operation as in previous versions of LabVIEW. [*Idea submitted by SteveP.*]
- **Get LV Class Default Value**—The **Get LV Class Default Value** VI now allows pseudopath values for the **class path** input.
- **Mean**—The **Mean** VI changed to a polymorphic VI and contains the new Mean (CDB) instance.
- **Complex Queue PtByPt** and **Data Queue PtByPt**—The **Complex Queue PtByPt** VI and the **Data Queue PtByPt** VI include an **initial queue element** input that specifies the initial element value in the data queues.
- **Build**—The **Build** VI changed to a polymorphic VI and contains the new Build (project reference) instance.
- **Read From Measurement File**—When you select **Text (LVM)** as the **File Format**, the **Read generic text files** checkbox no longer contains a checkmark by default.

New and Changed Classes, Properties, Methods, and Events

LabVIEW 2012 includes new VI Server classes, properties, methods, and events. Refer to the **LabVIEW 2012 Features and Changes»New VI Server Objects** topic on the **Contents** tab of the *LabVIEW Help* for a list of new classes, properties, methods, and events.

New and Changed Web Services

Web Services Management

NI Web-based Configuration and Monitoring includes the **Web Services Management** page. Use this page to undeploy Web services, view user permissions for individual Web services, pause and restart deployed Web services, and invoke Web methods. To access the **Web Services Management** page, right-click a LabVIEW RESTful build specification and select **Undeploy** from the shortcut menu.

Variable Web Services

LabVIEW Web services allow HTTP-capable clients to interact with VIs to exchange data. You use URLs and HTTP methods to transmit data to the Web service. LabVIEW 2012 includes a built-in Web service that enables you to access shared variables. The shared variable Web Service, nivariable, uses the Open Data Protocol (OData). Refer to the OData website at `www.odata.org` for information about OData. You can use the shared variable Web service to interact with and monitor shared variables that you create using LabVIEW.

Accomplishing Common Goals Using Project Templates and Sample Projects

LabVIEW 2012 provides project templates to help you create projects that use reliable designs and programming practices. To demonstrate how to adapt these project templates for authentic purposes, LabVIEW 2012 also includes a sample project based on each template. Select **File»Create Project** to browse the following templates and sample projects using the **Create Project** dialog box:

- **Simple State Machine**—A template that defines sections of code and the order in which they execute.
- **Finite Measurement**—A sample project that uses the Simple State Machine template to acquire a single measurement and log it to disk.
- **Queued Message Handler**—A template that monitors a user interface while allowing you to perform other tasks, such as data acquisition or logging.
- **Continuous Measurement and Logging**—A sample project that uses the Queued Message Handler to acquire measurements continuously and log them to disk.
- **Actor Framework**—A template for an application that consists of multiple independent tasks that communicate with one another.
- **Feedback Evaporative Cooler**—A sample project that uses the Actor Framework to cool a water tank, where the cooler is represented by either simulated or real hardware, controlled either manually or automatically, and viewed with one of two user interfaces.

Refer to the `Project Documentation` folder in each template or sample project for more information about the design and functionality of the project.

Features and Changes in Previous Versions of LabVIEW

To identify new features in each version of LabVIEW that released since your previous version, review the upgrade notes for those versions. To access these documents, refer to the National Instruments website at `ni.com/info` and enter the Info Code for the appropriate LabVIEW version from the following list:

- *LabVIEW 8.6 Upgrade Notes*—`upnote86`
- *LabVIEW 2009 Upgrade Notes*—`upnote9`
- *LabVIEW 2010 Upgrade Notes*—`upnote10`
- *LabVIEW 2011 Upgrade Notes*—`upnote11`

371780H-01 Jun12