LabVIEW™ 2017 Upgrade Notes

These upgrade notes describe the process of upgrading LabVIEW for Windows, OS X, and Linux to LabVIEW 2017. 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 2017

Contents

Upgrading to LabVIEW 2017

Although you can upgrade small applications to a new version of LabVIEW by installing the new version and then loading your VIs, NI 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 NI 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

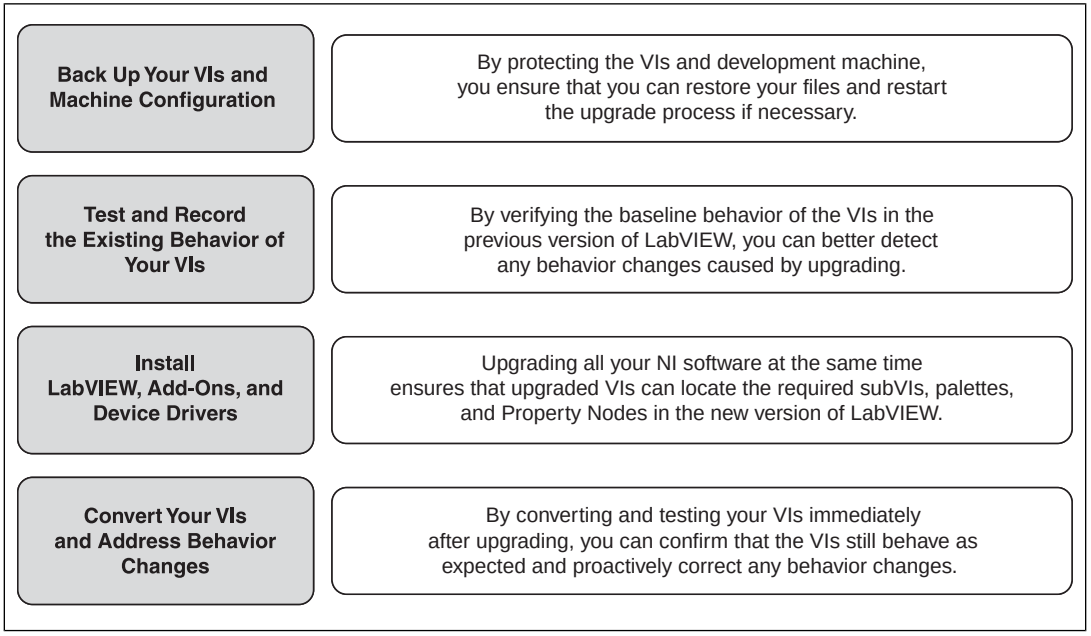

Figure 1.

Note To upgrade from LabVIEW 5.1 or earlier, you must first upgrade to an intermediate version of LabVIEW. Refer to the NI 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 2017, 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. 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, NI 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, differences between the previous version of LabVIEW and LabVIEW 2017 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:

- 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.

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 2017, copy files from the `labview\user.lib` directory in the previous version to the `labview\user.lib` directory in LabVIEW 2017.

4. Convert Your VIs and Address Behavior Changes

Mass compiling your VIs in LabVIEW 2017 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 2017. 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.

Note When you mass compile VIs that contain FPGA or real-time resources, the **Mass Compile** dialog box may report the VIs as non-executable VIs. To check for errors, you must open the VIs under the FPGA or RT target in a LabVIEW project with the required FPGA or real-time resources.

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

Differences between your previous version of LabVIEW and LabVIEW 2017 can occasionally cause some VIs to break if they use modified features. To quickly identify and fix broken VIs in LabVIEW 2017, 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 NI 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. Refer to the NI website at `ni.com/info` and enter the Info Code `analyzevi` to download these tests.
  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 2014 Upgrade VI Analyzer tests locate a potential behavior change, refer to the *Upgrading from LabVIEW 2013* 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 2017 Known Issues list—Lists bugs discovered before and throughout the release of LabVIEW 2017. Refer to the NI website at `ni.com/info` and enter the Info Code `lv2017ki` to access this list. Refer to the *Upgrade - Behavior Change* and *Upgrade - Migration* sections, if available, to identify workarounds for any bugs that may affect the behavior of upgraded VIs.
  - Module and toolkit documentation—Lists upgrade issues specific to some modules and toolkits, such as the LabVIEW FPGA Module and the LabVIEW Real-Time Module.
  - 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 2017.

- **Your own tests**—Perform the same tests on the VIs in LabVIEW 2017 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 2017»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 2017 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 2017.

Upgrading from LabVIEW 2012 or Earlier

Refer to the NI website at `ni.com/info` and enter the Info Code `upnote13` to access upgrade and compatibility issues you might encounter when you upgrade to LabVIEW 2017 from LabVIEW 2012 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 2013

You might encounter the following compatibility issues when you upgrade to LabVIEW 2017 from LabVIEW 2013. Refer to the *Upgrading from LabVIEW 2014*, *Upgrading from LabVIEW 2015*, and *Upgrading from LabVIEW 2016* sections of this document for information about other upgrade issues you might encounter.

Behavior Change in the String to Path Function

In LabVIEW 2014 and later, the String To Path function is case insensitive when reading any variation of the string `<Not A Path>` and always returns a constant value of <Not A Path>. For example, you can specify `<not a path>` or `<Not A Path>` in the **string** input, and in both cases, the function returns a constant value of <Not A Path>. Refer to the following table for more information about how the String to Path function behaves in previous versions of LabVIEW.

LabVIEW 2012 and 2013	LabVIEW 2011 and Earlier
Regardless of case, the String To Path function does not return a constant value of <Not A Path>. You can specify any variation of the string `<Not A Path>`, and the function returns a path to a directory named `<Not A Path>` instead of returning a constant value of <Not A Path>.	Like LabVIEW 2014 and later, the String To Path function is case insensitive and returns the constant value of <Not A Path> when you specify any variation of the string `<Not A Path>`. Whether you specify `<not a path>` or `<Not a Path>`, the function returns the constant value of <Not A Path>.

Reviewing and Updating Type Definitions

The **Review and Update from Type Def** shortcut menu item replaces the **Update from Type Def** shortcut menu item that appears in LabVIEW 2013 and earlier.

Deprecated VIs, Functions, and Nodes

LabVIEW 2014 and later do not support the following VIs, functions, and nodes.

Apple Event VIs

(OS X) LabVIEW 2014 and later no longer support Apple Event VIs. Instead, use the Run AppleScript Code VI on the Libraries & Executables palette to communicate with OS X applications external to

LabVIEW. If you attempt to load a VI that contains any of the following Apple Event VIs, LabVIEW may generate errors and be unable to run the VI:

- AESend Do Script
- AESend Finder Open
- AESend Open
- AESend Open Document
- AESend Print Document
- AESend Quit Application
- Get Target ID
- AESend Abort
- AESend Close
- AESend Open, Run, Close
- AESend Run
- AESend VI Active?
- AECreate Comp Descriptor
- AECreate Descriptor List
- AECreate Logical Descriptor
- AECreate Object Specifier
- AECreate Range Descriptor
- AECreate Record
- AESend
- Make Alias

Actor Framework VIs

LabVIEW 2014 and later do not support the Actor:Launch Actor VI. Use the Actor:Launch Root Actor VI or Actor:Launch Nested Actor VI instead.

In Port and Out Port VIs

LabVIEW 2014 and later do not support the In Port and Out Port VIs.

Deprecated Properties, Methods, and Events

LabVIEW 2014 and later do not support the Get VI:Old Help Info method of the Application class. Instead, use the Get VI:Help Info method to return help information from the **Documentation** page of the **VI Properties** dialog box for a specified VI.

Upgrading from LabVIEW 2014

You might encounter the following compatibility issues when you upgrade to LabVIEW 2017 from LabVIEW 2014. Refer to the *Upgrading from LabVIEW 2015* and *Upgrading from LabVIEW 2016* sections of this document for information about other upgrade issues you might encounter.

Identifying Buffer Allocations in LabVIEW Applications

LabVIEW 2014 Service Pack 1 and later include the **Profile Buffer Allocations** window to identify and analyze buffer allocations in a LabVIEW application. Select **Tools»Profile»Profile Buffer Allocations** to display this window.

Hyperlinks in Free Labels

LabVIEW 2015 and later detect URLs in free labels and converts them to hyperlinks underlined in blue text. LabVIEW does not automatically convert URLs in free labels to hyperlinks when you upgrade

from LabVIEW 2014 or earlier. To enable hyperlinks in front panel labels, right-click the free label and select **Enable Hyperlinks** in the shortcut menu. You cannot disable hyperlinks in block diagram labels.

Deprecated VIs, Functions, and Nodes

LabVIEW 2015 and later do not support the following VIs, functions, and nodes.

- **Read From Spreadsheet File**—Use the Read Delimited Spreadsheet VI instead.
- **Write To Spreadsheet File**—Use the Write Delimited Spreadsheet VI instead.

Upgrading from LabVIEW 2015

You might encounter the following compatibility issue when you upgrade to LabVIEW 2017 from LabVIEW 2015. Refer to the *Upgrading from LabVIEW 2016* section of this document for information about other upgrade issues you might encounter.

In LabVIEW 2016 and later, the **Quick Drop Configuration** dialog box contains a default list of shortcuts for front panel and block diagram objects. Shortcuts you created in LabVIEW 2015 or earlier do not automatically migrate to the list of shortcuts in LabVIEW 2016 and later.

Upgrading from LabVIEW 2016

You might encounter the following compatibility issue when you upgrade to LabVIEW 2017 from LabVIEW 2016.

Behavior Change in the Actor Framework VIs

In LabVIEW 2016 and earlier, when a nested actor fails to launch because of an error in the Pre-Launch Init method, the nested actor returns an error and sends a Last Ack message that contains the error to its caller actor. In LabVIEW 2017, the nested actor returns an error without sending a Last Ack message to its caller actor.

LabVIEW 2017 Features and Changes

The Idea Exchange icon denotes a new feature idea that originates from a product feedback suggestion on the NI Idea Exchange discussion forums. Refer to the NI 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 2017.

Reduced VI Load and Compile Time

(Windows) For LabVIEW 2017, NI upgraded to a more aggressive compiler for building both the LabVIEW development environment and the LabVIEW Run-Time Engine. This upgrade reduced aggregate VI load time and VI compile time.

Maintaining Wire Connections When Moving Objects

LabVIEW 2017 automatically maintains wire connectivity when you move objects in and out of structures on the block diagram. When an object moving in or out of a structure is connected to an object in the structure, LabVIEW creates or removes tunnels to maintain wire connectivity. You can toggle automatic wire connectivity when moving objects by pressing the <W> key.

Malleable VIs

LabVIEW 2017 includes malleable VIs (`.vim`) that are inlined into their calling VIs and can adapt each terminal to its corresponding input data type. With malleable VIs, you create a VI to perform the same operation on any acceptable data type instead of saving a separate copy of the VI for each data type.

A malleable VI is similar to a polymorphic VI but is more flexible when determining which data types are acceptable. A polymorphic VI uses a predefined list of acceptable data types. A malleable VI computes whether a data type is acceptable by implementation.

Malleable VIs use the `.vim` file extension. You can create a malleable VI by selecting **File»New** and selecting **Malleable VI** from the **New** dialog box. You can convert an existing VI into a malleable VI by saving the file with the `.vim` file extension.

Note You can convert only standard VIs into malleable VIs. You cannot convert polymorphic VIs, global VIs, or XControl abilities into malleable VIs.

Built-In Malleable VIs

LabVIEW provides the following malleable VIs for use in your applications. The icons of the built-in malleable VIs have orange backgrounds.

- Array palette
  - **Decrement Array Element**—Subtracts 1 from the specified element of a 1D array. If the array is an array of timestamps, this VI decrements the element by one second.
  - **Increment Array Element**—Adds 1 to the specified element of a 1D array. If the array is an array of timestamps, this VI increments the element by one second.
  - **Shuffle 1D Array**—Rearranges the elements of a 1D array in a pseudorandom order.
  - **Shuffle 2D Array**—Rearranges the elements of a 2D array in a pseudorandom order.
  - **Sort 2D Array**—Rearranges the rows or columns of a 2D array by sorting the elements in the specified column or row in ascending order.
- Comparison palette
  - **Is Value Changed**—Returns TRUE if this is the first call of this VI or if the input value is different from the value when this VI was last called.
- Conversion palette
  - **Number To Enum**—Looks for an enum value that matches the specified number and returns the corresponding enum item.
- Timing palette
  - **Stall Data Flow**—Delays the data flow of the wire for a specified period of time.

Refer to the **Fundamentals»Creating VIs and SubVIs»Concepts»Creating Modular Code»Malleable VIs** topic on the **Contents** tab in the *LabVIEW Help* for information about malleable VIs.

Refer to the `labview\examples\Malleable VIs\Basics\Malleable VIs Basics.lvproj` for an example of using malleable VIs.

[Idea submitted by NI Discussion Forums member DanyAllard.]

New and Changed VIs and Functions

LabVIEW 2017 includes the following new and changed VIs and function:

Read-Only Access for Data Value References

The Data Value Reference Read / Write Element border node of the In Place Element structure can allow read-only access to a data value reference. Right-click the border node on the right of the structure and select **Allow Parallel Read-Only Access**. When the border node on the right is unwired, LabVIEW allows multiple, concurrent read-only operations and does not modify the data value reference.

New Channel Templates

LabVIEW 2017 includes the Event Messenger channel template. Use this channel to transfer data from multiple writers to one or more Event structures. Each write operation to the channel triggers an event. The Event Messenger channel allows the channel syntax to combine with the event syntax that controls your user interface events and generated events. Refer to the `labview\examples\Channels\Event Messenger\Channel - Event Messenger.lvproj` for an example of using the Event Messenger channel.

New and Changed Classes, Properties, Methods, and Events

LabVIEW 2017 includes changes to the Get VI Dependencies (Names and Paths) method. The **Keep Express VIs?** parameter is renamed to **Keep Express and Malleable VIs?**. If **Keep Express and Malleable VIs?** is FALSE (default), LabVIEW returns the names of the hidden instance VIs that underlie the Express VIs and malleable VIs. If TRUE, LabVIEW returns the Express VIs and malleable VIs as dependencies. If you want edit-time dependencies, set **Keep Express and Malleable VIs?** to TRUE. If you want run-time dependencies, set **Keep Express and Malleable VIs?** to FALSE. Regardless of this setting, LabVIEW includes the subVIs of the instance VIs as dependencies of the referenced VI.

Application Builder Enhancements

LabVIEW 2017 includes the following enhancements to the LabVIEW Application Builder and build specifications:

Backward Compatibility of the LabVIEW Run-Time Engine

In previous versions of LabVIEW, you cannot load and run binaries and VIs built in older versions of LabVIEW without recompilation. Starting from 2017, LabVIEW supports backward compatibility for the LabVIEW Run-Time Engine. For example, versions of LabVIEW later than 2017 can load binaries and VIs built with LabVIEW 2017 without recompiling. This improvement applies to stand-alone applications (EXEs), shared libraries (DLLs), and packed project libraries.

To enable binaries to be backward compatible, place a checkmark in the following checkbox on the **Advanced** page of the specific dialog box depending on your build specification:

Build Specification	Dialog Box	Checkbox
Stand-alone application (EXE)	**Application Properties**	**Allow future versions of the LabVIEW Runtime to run this application**
Packed project library	**Packed Library Properties**	**Allow future versions of LabVIEW to load this packed library**
Shared library (DLL)	**Shared Library Properties**	**Allow future versions of LabVIEW to load this shared library**

LabVIEW enables these options by default for build specifications you create in LabVIEW 2017 and later. You can disable these options to bind a build specification to a specific version of LabVIEW. Disabling these options prevents any changes to the performance profiles and helps you avoid unexpected problems resulting from compiler upgrades. For real-time applications, these options do not appear in the dialog boxes but the functionality is enabled by default.

Improvements to Calls between LabVIEW and Other Languages

In LabVIEW 2017, the performance and stability of LabVIEW-built shared libraries (DLLs) are improved significantly, specifically for calls to LabVIEW-built DLLs from LabVIEW and other languages. For example, calls to LabVIEW-built DLLs from a C-language application now run in a multithreaded

execution system. The improvements also prevent some potential deadlocks and atomicity violations while calling LabVIEW-built DLLs from LabVIEW.

To use this functionality, place a checkmark in the **Execute VIs in private execution system** checkbox on the **Advanced** page of the **Shared Library Properties** dialog box. By default, this option is enabled for new build specifications. This option is disabled for build specifications migrated from LabVIEW 2016 and earlier to prevent unintended changes in behavior. For example, disabling this option prevents shared libraries that rely on single-threaded execution to execute in a multithreaded execution system, when LabVIEW-built shared libraries are called from a non-LabVIEW application. **(NI Linux Real-Time)** This option is disabled by default for Linux RT targets because of potential performance jitters.

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 NI website at `ni.com/info` and enter the Info Code for the appropriate LabVIEW version from the following list:

- *LabVIEW 2013 Upgrade Notes*—`upnote13`
- *LabVIEW 2014 Upgrade Notes*—`upnote14`
- *LabVIEW 2015 Upgrade Notes*—`upnote15`
- *LabVIEW 2016 Upgrade Notes*—`upnote16`

371780N-01 Mar17