Best Practices for Managing NI LabVIEW Applications Using the Project Explorer

Publish Date: May 15, 2019

Overview

(http://www.ni.com/white-paper/document/tut-7117)This article is part of a series on software engineering practices and tools for large application development in LabVIEW.

Click here (http://www.ni.com/white-paper/document/tut-7117) to view the list of other articles

This article examines the LabVIEW Project and provides recommendations for how to manage and organize your LabVIEW applications. Use these recommendations to establish guidelines and procedures before beginning development to ensure that your application scales for large numbers of VIs and for multiple developers.

Table of Contents

1. Defining and Identifying Application Files

Establishing guidelines for storing and managing files requires foresight into how the application will be structured, how functionality will be divided, and the types of files beyond source code that will be important to keep track of. Devote time to making decisions about how functionality will be divided among code and to working with developers on file storage locations and the additional files or resources they will need to function properly. Use this information in the next section to determine the criteria by which you group files together on disk.

2. Organizing Files on Disk

File organization on disk should not be an afterthought. Poor planning for large applications leads to additional time spent moving and renaming files during development. When dealing with large numbers of files, remember that these operations can pose considerable risk to the integrity of links within LabVIEW and, therefore, to the behavior of your application. Establishing these practices as early as possible mitigates the risk of moving large sets of files at a later date, and you can help ensure that developers can easily find files and determine where to save new files.

Many software developers already have practices and systems in place to determine where files should be stored. Ultimately, there is a large combination of practices and structures that are legitimate options, but the following are established common practices that have proven to scale well for large applications. Storing files within a single root directory ensures that everything you need is easily accessible, and enforcing a standard naming convention helps users know where to find files and where to place new files. Folders on the hard drive are commonly used to group or categorize files and thereby separate subVIs from callers. Conversely, flat directory structures make it difficult to find files or to locate the top-level VI and are not recommended even for small numbers of VIs. The criteria typically used for grouping these files are a combination of the file functionality, type, and hierarchical layer in the application. In fact, organization on disk should be a physical manifestation of the relationship of files and code in the application.

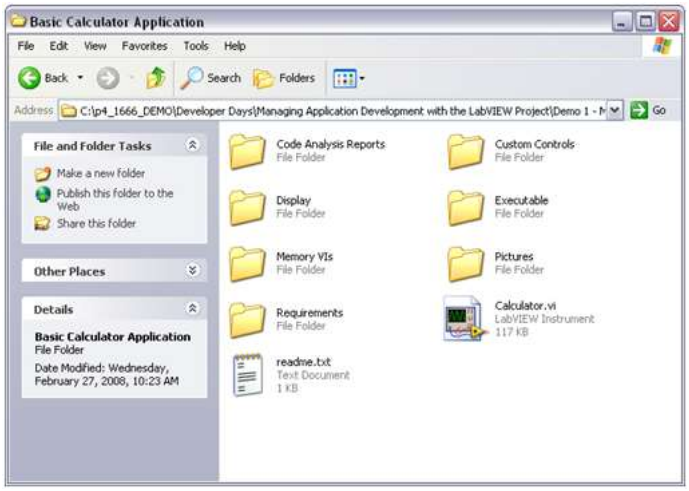

Figure 1 - This screen-shot demonstrates an example application that was organized on disk without the use of the LabVIEW Project Explorer.

Figure 1 demonstrates concepts such as organizing all files within a root directory, separating the top-level VI from dependencies, and the various criteria by which to 'bucket' files into folders.

Recommendations

- Store all files within a single root directory
- Divide your application into logical pieces of manageable size
- Use logical and descriptive naming conventions
- Separate the top-level VIs from other source code
- Begin with a high-level block diagram that includes the main components of your application (for example, the block diagram could include separate frameworks for configuration, acquisition, analysis, data display, data logging, and error handling)
- Group or “bucket” files according to predetermined criteria

Special Consideration for Dynamic Dependencies and Shared Code

Dynamically loaded files pose an additional challenge that often warrants special consideration. Because these files are not statically linked by any callers in the application, they can be easily misplaced or forgotten when moving locations or in any way changing paths to files.

To minimize the difficulty involved with ensuring you’ve kept all dynamically accessed files in the correct location, you can group them in a separate folder and refer to them using relative paths. If it is necessary to move or distribute the application, you can be sure you have the files you need just by including the folder.

When large applications share code, it is important to be mindful of where these files are stored and that changes may inadvertently affect other callers. Code reuse is common and encouraged to speed development by helping you take advantage of work that has already been completed. Applications that are similar may require the same functionality, or different branches of code may refer to the same set of common VIs. Referring to the same copy of code often requires that these files be located outside of the root directory of at least one application. However, if you plan to make changes to these common VIs, make local copies for testing and later integration.

You also can use source code control providers to track multiple versions of software and compare the changes that are made between multiple iterations. For more information on this topic, see NI LabVIEW Development for Team-Based Projects (http://www.ni.com/tutorial/11537/en/). You also can read more about Source Control in the LabVIEW Help (http://zone.ni.com/reference/en-XX/help/371361R-01/lvconcepts/using_source_code/).

Recommendations

- Keep dynamically loaded dependencies in a common folder
- Use relative paths to reference dynamic dependencies
- Minimize changes to code shared by applications
- Dedicate time to managing and integrating changes to shared code
- Use source code control

Summary of File Organization on Disk

Organizing applications on disk is fundamental to ensuring that development can scale as the size and number of VIs grow. However, consider this as the first step in a process. There are still many inherent challenges that you can mitigate using more sophisticated practices and tools such as the LabVIEW Project Explorer.

3. The LabVIEW Project Explorer

The Project Explorer was introduced in LabVIEW 8.0 to provide developers with a system-level view of the files they need for an application. The goal of the LabVIEW Project Explorer is to help developers easily find and organize files from within the development environment and provide added functionality that addresses the challenges of managing and developing large LabVIEW applications. Since the initial release of the LabVIEW Project Explorer, National Instruments has incorporated several new features as a result of customer feedback. To learn more about the latest version of the Project Explorer window, view the Managing a Project in LabVIEW (http://zone.ni.com/reference/en-XX/help/371361R-01/lvconcepts/using_labview_projects/) topic in the LabVIEW Help.

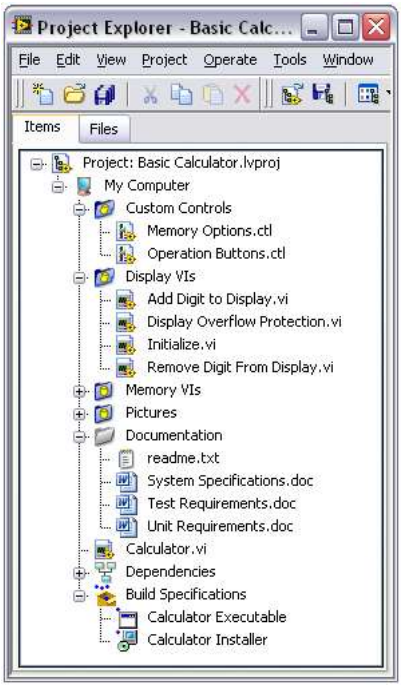

Figure 2 - This Project demonstrates how the same application we saw in Figure 1 can be organized in the LabVIEW Project Explorer.

Use LabVIEW Projects to group together LabVIEW files and other files, create build specifications, and deploy or download files to targets. When you save a LabVIEW Project, LabVIEW creates a .lvproj file, which includes references to files on disk, variables, hyperlinks, configuration information, build information, deployment information, and so on.

The LabVIEW Project Explorer does not reproduce or copy your files to new locations on disk. It provides a new method for displaying files on disk and offers shortcuts to the locations of the files on disk. With the LabVIEW Project Explorer, you can:

- Easily access and navigate files without leaving LabVIEW
- Customize and filter how files are organized
- Preserve links when moving or renaming files on disk
- Prevent, detect, and resolve incorrect links
- Manage code for specific hardware targets[1]
- Manage build configurations[1]
- Integrate with source code control providers[1]

[1]These are not covered in this article.

Note that some of the functionality described in this article is available only in LabVIEW 8.5 or later. For a list of new features in the latest version of LabVIEW, view the upgrade notes or visit http://www.ni.com/labview/whatsnew/upgrade (http://www.ni.com/labview/whatsnew/upgrade).

Managing Your Files in the Project Explorer

There are two pages for viewing files in the LabVIEW Project: the Items page and the Files page.

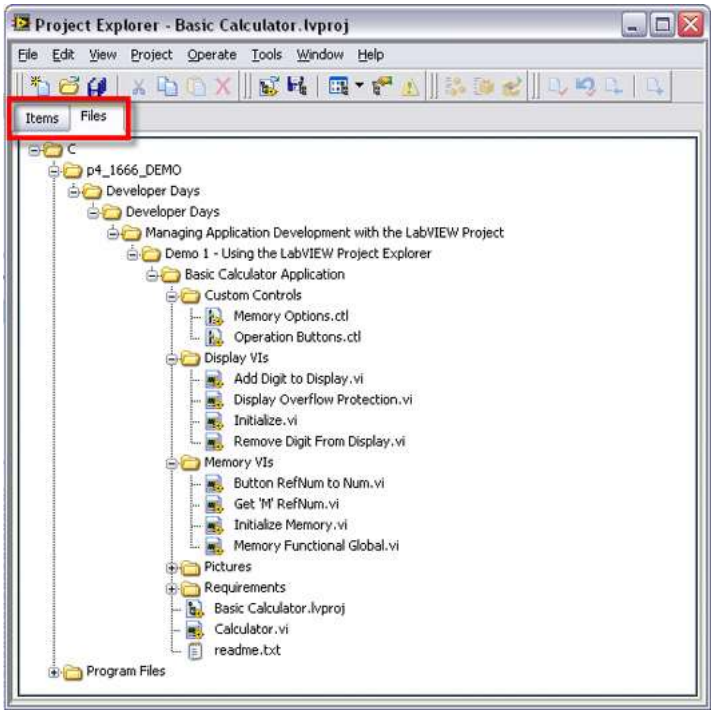

Figure 3 - The files page shows a filtered view of all items in the Project as they are located on disk and allows users to perform file operations such as copy and rename

The default view is the Items page, which provides a tree view of items you have added to the LabVIEW Project and groups them by hardware target. You can create folders in this view to either customize how files are organized or to synchronize to a specific location on disk. This is where you likely will spend most of your time. You cannot delete files from disk from the Items page – this is designed to protect users from accidentally deleting code. You can delete items from the Files page only by right-clicking on them and selecting “Delete from Disk.”

The Files page shows where items that have a corresponding file on disk are physically located. Use this view to perform the same file operations you would normally perform from within your system file browser, such as moving, copying, or deleting. Because LabVIEW is aware of the changes, it is able to update callers and preserve links when changes are made.

You can customize the hierarchy of files in the LabVIEW Project Explorer without affecting the layout of files on disk; however, it is recommended that the organization in the LabVIEW Project reflect the hierarchy you’ve set up on disk as closely as possible.

Using Folders in the LabVIEW Project

You can organize the contents of your LabVIEW Project using two types of folders: virtual folders and autopopulating folders. With virtual folders, you can customize the organization of project items without making any modifications on disk.

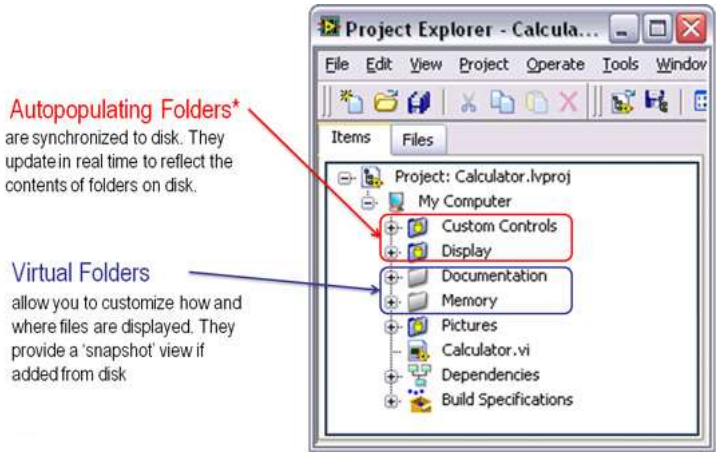

Figure 4 - Users can select from auto-populating or virtual folders when organizing files in the LabVIEW Project Explorer

Virtual folders are ideal for scenarios in which you want to change the organization of files in LabVIEW without making any modifications to their storage locations on disk. If you add a folder on disk to the LabVIEW Project Explorer as a virtual folder, it is represented by a snapshot view at the time you added it. If you make any changes afterward, including the addition of new files, these changes do not appear in the LabVIEW Project. However, you can make changes manually by dragging new files into the LabVIEW Project or temporarily converting the virtual folder to an autopopulating folder.

With virtual folders, you can group files from two or more separate locations on disk in the LabVIEW Project Explorer. They are also useful for filtering the information displayed in the LabVIEW Project Explorer to help you determine if files are grouped with other items you don't need.

NI introduced autopopulating folders in LabVIEW 8.5 to automatically reflect the contents of folders on disk. Autopopulating folders are synchronized to a physical folder, which means that they always display any changes or modifications made to this location outside of the LabVIEW development environment. Use autopopulating folders whenever possible to preserve the disk hierarchy within the LabVIEW Project Explorer. Autopopulating folders are not recommended when managing Project Libraries or LabVIEW Classes or when attempting to resolve large numbers of cross-links.

Recommendations

Use autopopulating and virtual folders to enforce organization and structure for your application files in the LabVIEW Project Explorer. Autopopulating folders are ideal for preserving and maintaining the structure on disk, but there are situations in which you either cannot use them or need customization. In these cases, use virtual folders to customize the organization of files in the LabVIEW Project Explorer.

Dependencies

LabVIEW automatically loads all statically linked subVIs into memory when you open the calling VI. The Dependencies section of the LabVIEW Project Explorer is automatically populated with these same subVIs if they have not been added to the LabVIEW Project when the calling VI is loaded into the LabVIEW Project Explorer. Beginning in LabVIEW 8.5, user dependencies are separated from the contents of vi.lib to make it easier to identify which files were written by the application's developers.

Adding files to a LabVIEW Project indicates to LabVIEW which files and resources are intended for use in an application. Check the dependencies section to make sure you've added everything you need to your LabVIEW Project – if you have not done so, remaining files are listed in this section. This is an important practice for ensuring you're using the copy of a subVI you intended.

A VI in your LabVIEW Project may become incorrectly linked to the wrong subVI, in which case adding the copy of the subVI you intend to use to the LabVIEW Project signals LabVIEW that one or more callers may be incorrectly linked to a different subVI with the same name. This scenario is often referred to as cross-linking.

4. Cross-Linking

LabVIEW opens subVIs from locations stored in the caller using relative paths. If LabVIEW does not find a subVI at the expected location, it searches for the subVI by name. LabVIEW loads the first subVI it finds by that name into memory and informs the user about the modified links. Because multiple VIs with different functionalities may share a common name (such as initialize.vi), or two versions of the same VI may exist in separate locations, LabVIEW may link to the incorrect file.

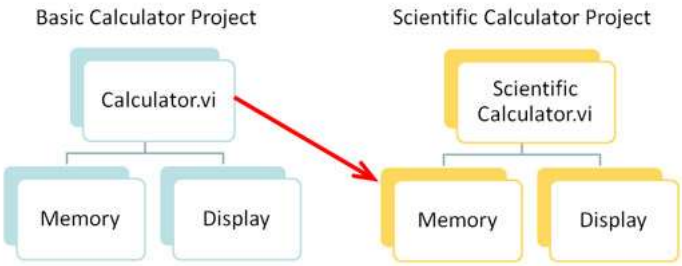

Figure 5 - High level illustration of cross-linking

Preventing Cross-Linking

Cross-linking commonly occurs as a result of moving or copying large sets of files to reproduce a hierarchy of code in a new disk location. If relative paths change, links may become corrupt and your application may unknowingly become linked to the incorrect set of dependencies.

LabVIEW cannot load two different VIs into memory with the same name. Therefore, opening two applications that reference separate subVIs with the same name means that one executes using the incorrect dependency.

Recommendations

There are several measures you can take to prevent cross-linking:

- Avoid working on multiple applications with common dependencies on the same machine simultaneously
- Use source code control applications to specify exactly where files should be located and to preserve correctly linked callers
- Avoid moving or copying segments of your application
- Add all your files to a LabVIEW Project to indicate which resources you intend to use
- Avoid use of common names like initialize.vi
- Qualify common names using Project Libraries for namespacing

Identifying Conflicts

Ensuring correct linkage is a common challenge across all programming languages. Careful practices can minimize the occurrence of cross-links and provide backups of correctly linked files, but it's still a big concern among vendors developing large applications. The LabVIEW Project in LabVIEW 8.5 introduced several new features targeted at helping developers detect and resolve cross-links.

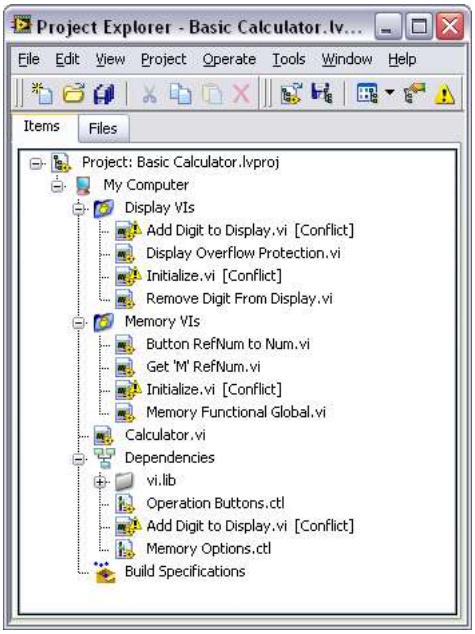

Figure 6 - LabVIEW is able to automatically warn the user of conflicts

Perhaps the most beneficial feature related to cross-linking is that the LabVIEW Project determines if a VI is calling a subVI with the same name as a file you've added to the LabVIEW Project. LabVIEW notifies you with a yellow warning indicator on the file's icon, at which point there are several methods for finding more information about the files to determine which file is the correct version:

- Right-click on the .lvproj file in the LabVIEW Project Explorer and select "Find Items with no Callers." If you have any unlinked VIs besides your top-level VI, they are listed here.
- Right-click on a VI and select **Find >> Callers** to identify which VI is calling it or if it does not have any callers.
- In the menu toolbar, select **Project >> Show Item Paths** to see the location on disk of all files in the Items page.
- Click on the "Resolve Conflicts.." toolbar icon to see detailed information about all cross-linked VIs.

Once you have identified cross-links and determined how to fix the links, you can make the corrections manually or use LabVIEW Project Explorer tools to walk through the process.

Resolving Conflicts

There are multiple methods for resolving conflicts after LabVIEW had detected them. Following the recommendations below helps ensure that you are able to preserve links to subVIs while making the required changes.

Recommendations

- If there are conflicts inside an autopopulating folder, you need to stop autopopulating or resolve the conflicts by renaming items in the Files page.
- If one or more callers have become incorrectly linked to a subVI outside of the LabVIEW Project Explorer, you need to relink these VIs to the appropriate subVI in the LabVIEW Project. Right-click a conflicting VI in the LabVIEW Project Explorer window and select "Replace" from the shortcut menu to choose the correct subVI on disk. Repeat this process for any remaining conflicts.
- If callers are referencing multiple items with the same name but only one item exists on disk, you can right-click a conflicting item and select "Replace with Item Found by Project" from the shortcut menu.
- If multiple items with the same name have been added to the LabVIEW Project and you want to use both in your application, rename them or add them to separate Project Libraries to qualify their names.

You can automate these recommendations with the "Resolve Conflicts" dialog box. Select **Project » Resolve Conflicts** from the LabVIEW Project Explorer window to display the "Resolve Conflicts" dialog box. You also can click the "Resolve Conflicts" toolbar button or right-click a conflicting item in the LabVIEW Project Explorer window and select "Resolve Conflicts" from the shortcut menu to display this dialog box.

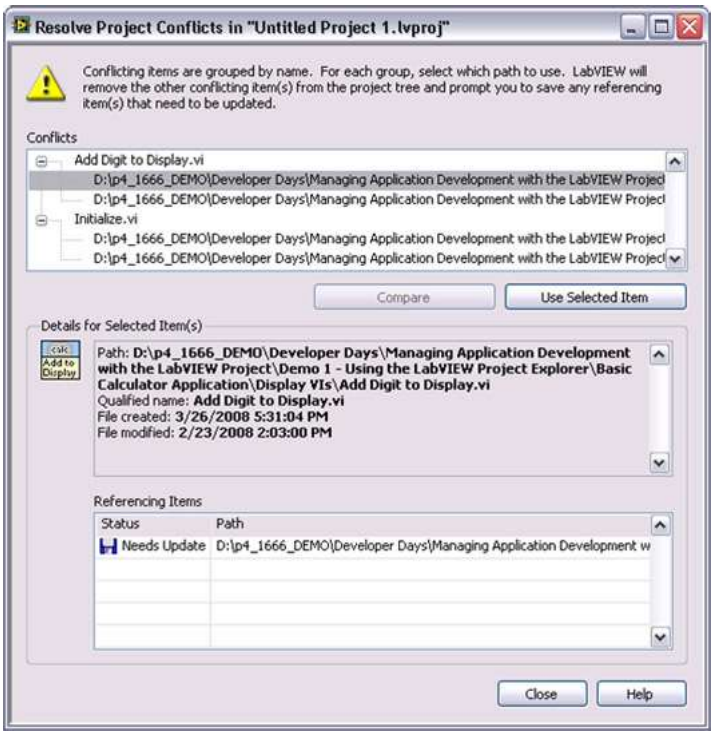

Figure 7 - Resolve conflicts dialog box prompts users to select the correct subVI

This dialog box includes the following components:

- Conflicts – Displays the project conflicts by qualified name.
- Compare – Compares two conflicting VIs.
- Use Selected Item – Resolves the conflicts when you select an item to use. If the conflict list involves only top-level items, LabVIEW removes all items in the conflict list from the LabVIEW Project except the item you select. LabVIEW cannot modify autopopulated folders in the “Resolve Project Conflicts” dialog box. You can manually resolve these items or disable the autopopulating folder by right-clicking the folder and selecting “Stop Auto-populating” from the shortcut menu.

Use this dialog box to resolve conflicts by redirecting the conflicting items to call dependent items from the correct path.

Summary

With the recommendations in this article, you can establish development and application management guidelines that ensure you can scale for large application development. Remember to give proper consideration to the organization of files on disk and establish criteria for sorting files. Use the LabVIEW Project Explorer for a system-level view of your application to easily access and navigate files and to take full advantage of tools for detecting cross-linking.

5. Related Resources

Software Engineering Resources (http://www.ni.com/white-paper/document/tut-7117)

- Prevent VI Cross-Linking with the Enhanced LabVIEW Project Explorer (http://www.ni.com/white-paper/document/tut-6200)
- NI LabVIEW Development for Team-Based Projects (http://www.ni.com/tutorial/11537/en/)
- Software Configuration Management and LabVIEW (http://www.ni.com/white-paper/document/tut-4114)
- LabVIEW Help: Sorting Items in a Project (http://zone.ni.com/reference/en-XX/help/371361R-01/lvhowto/organizing_items_project/)