LabVIEW Developer Days

Build Code. Form Communities. Gain Confidence.

NATIONAL INSTRUMENTS™

The Essentials of File Management with LabVIEW

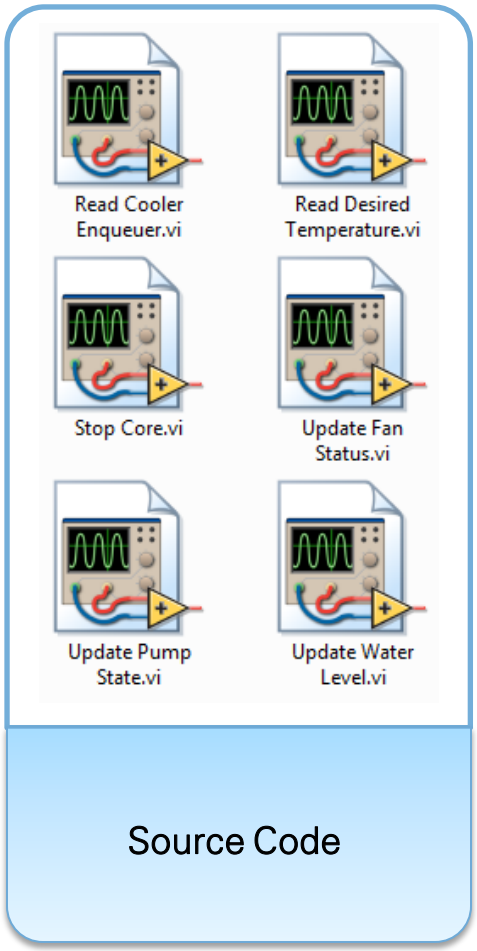
Session Goals

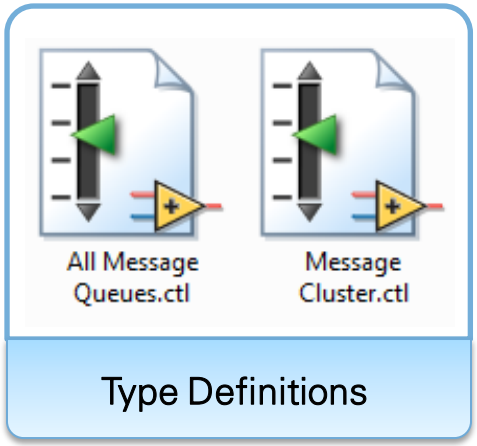
This session focuses on best practices for:

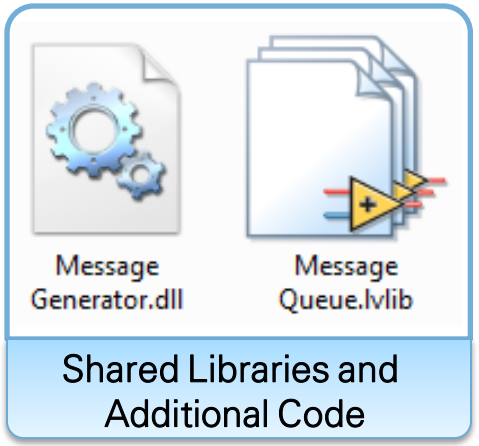
Organizing and Managing LabVIEW Applications
Managing a Code Base with Source Code Control
Building and Distributing Reuse Libraries

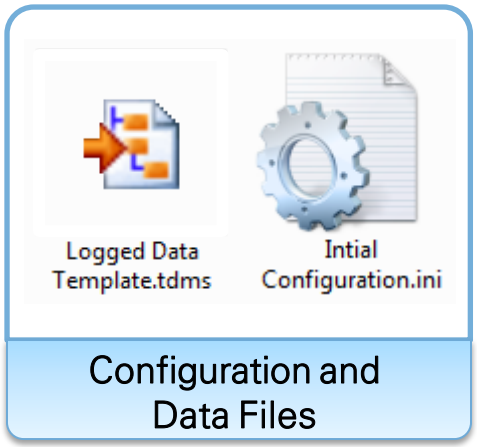
Organizing and Managing LabVIEW Applications

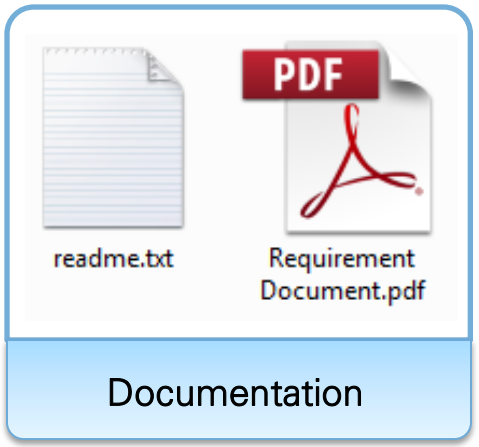
Defining a LabVIEW Application









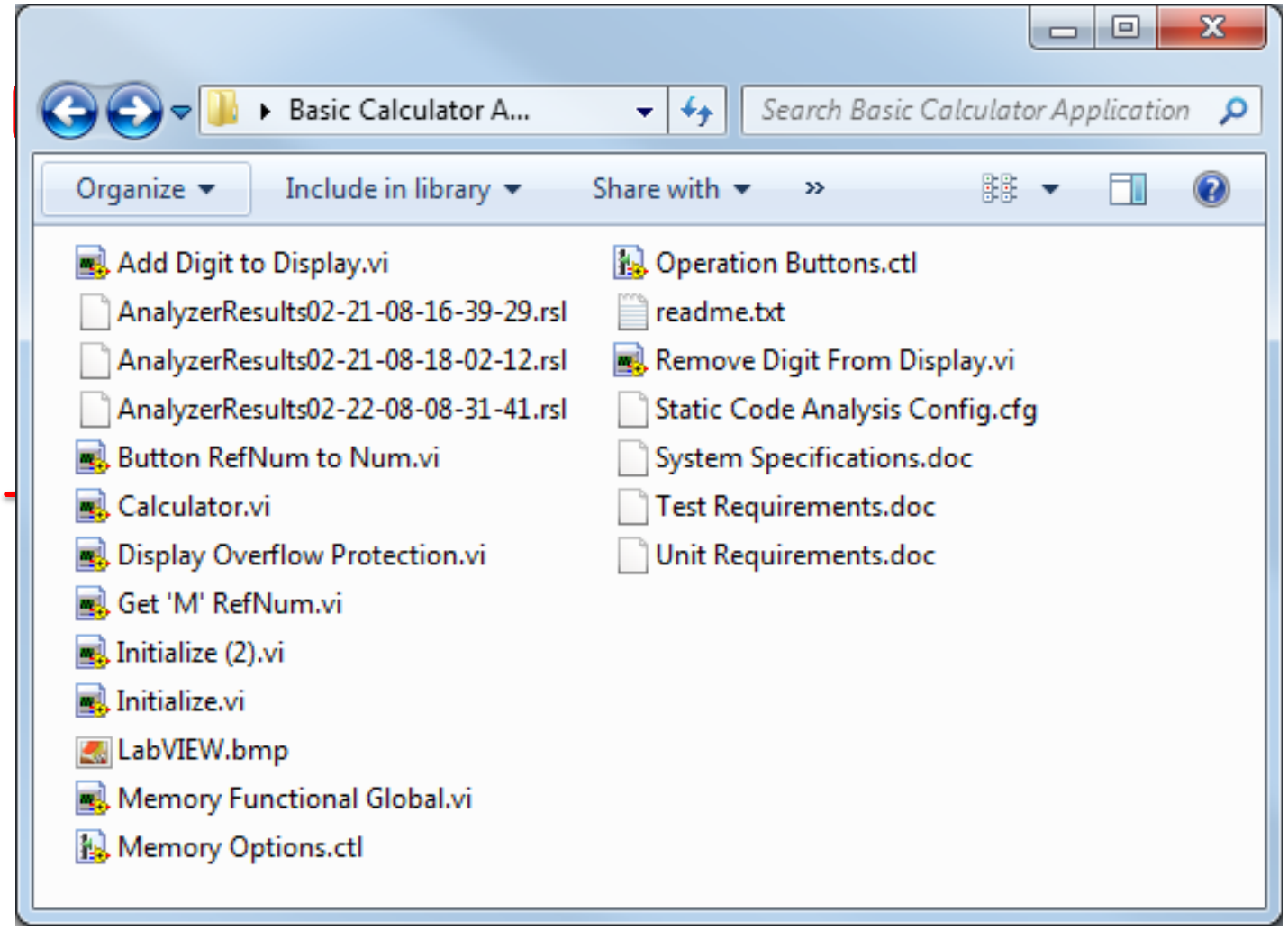


NATIONAL INSTRUMENTS

Recommendations for Organizing Files

Single Root Directory

Group related files using folders



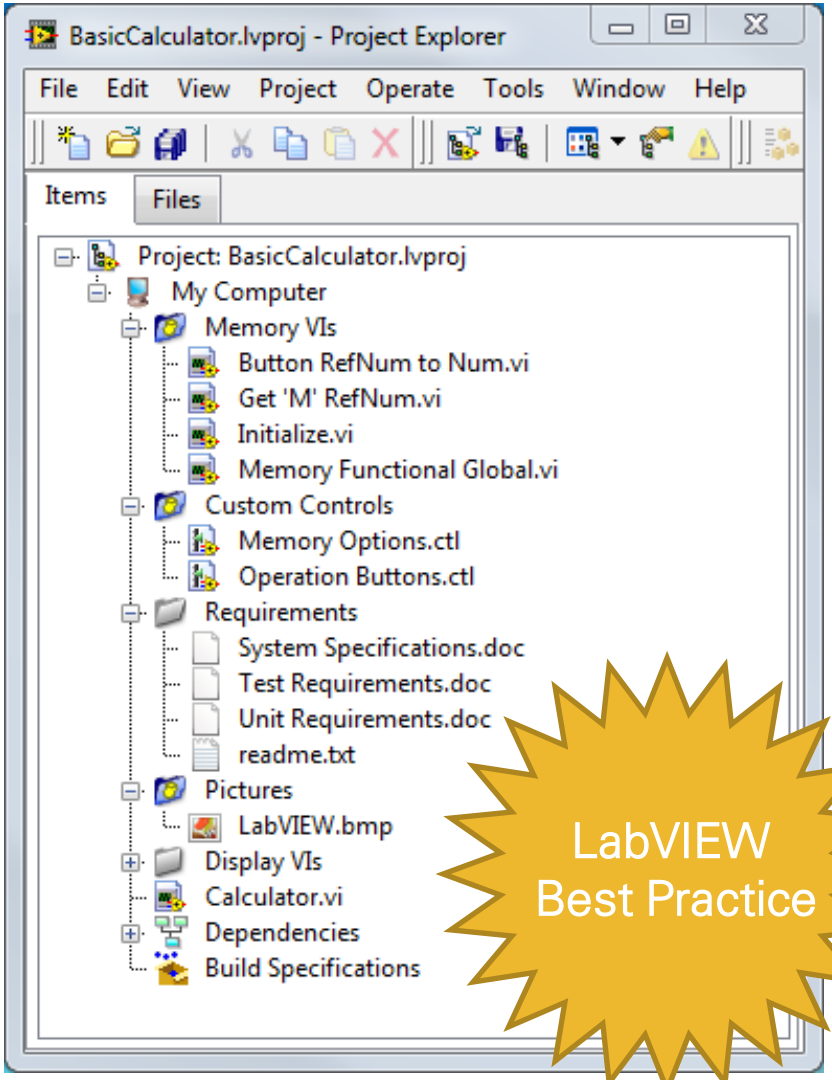
NATIONAL INSTRUMENTS

The Project Explorer

A tool to **improve developer efficiency** by providing:

- Easy file navigation
- Smart linking when moving files
- Integration with Application Builder
- Deploy code to LabVIEW targets
- Access to source code control

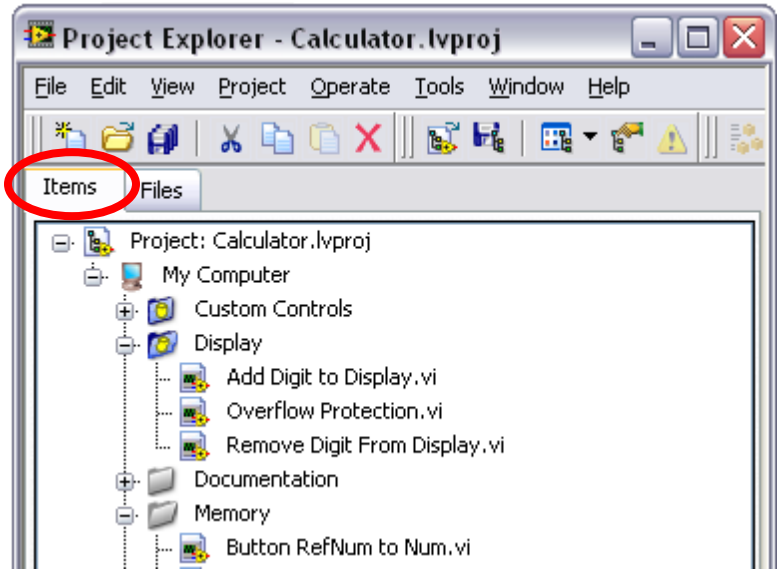
However, the project explorer <u>does not</u> duplicate files.



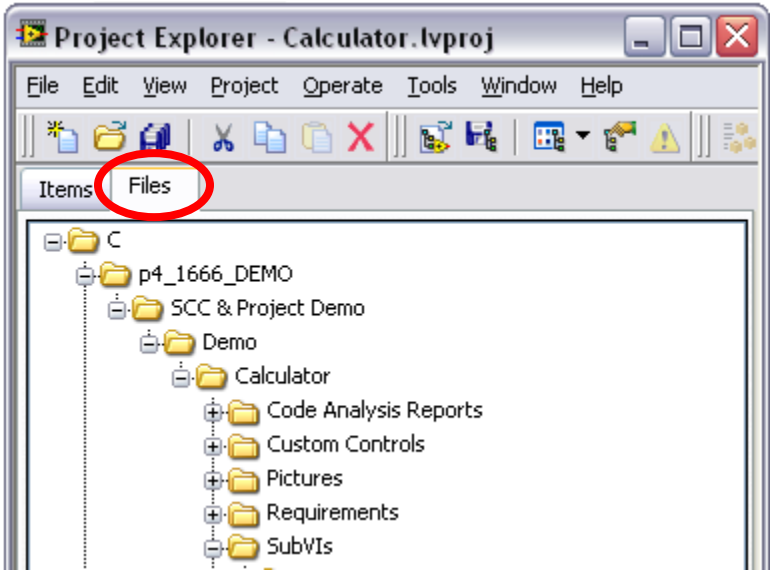
NATIONAL INSTRUMENTS

Improved File Management

Items View



Files View



If you move a file using the Files View, LabVIEW will be aware of the change and will update callers and preserve links automatically.

NATIONAL INSTRUMENTS

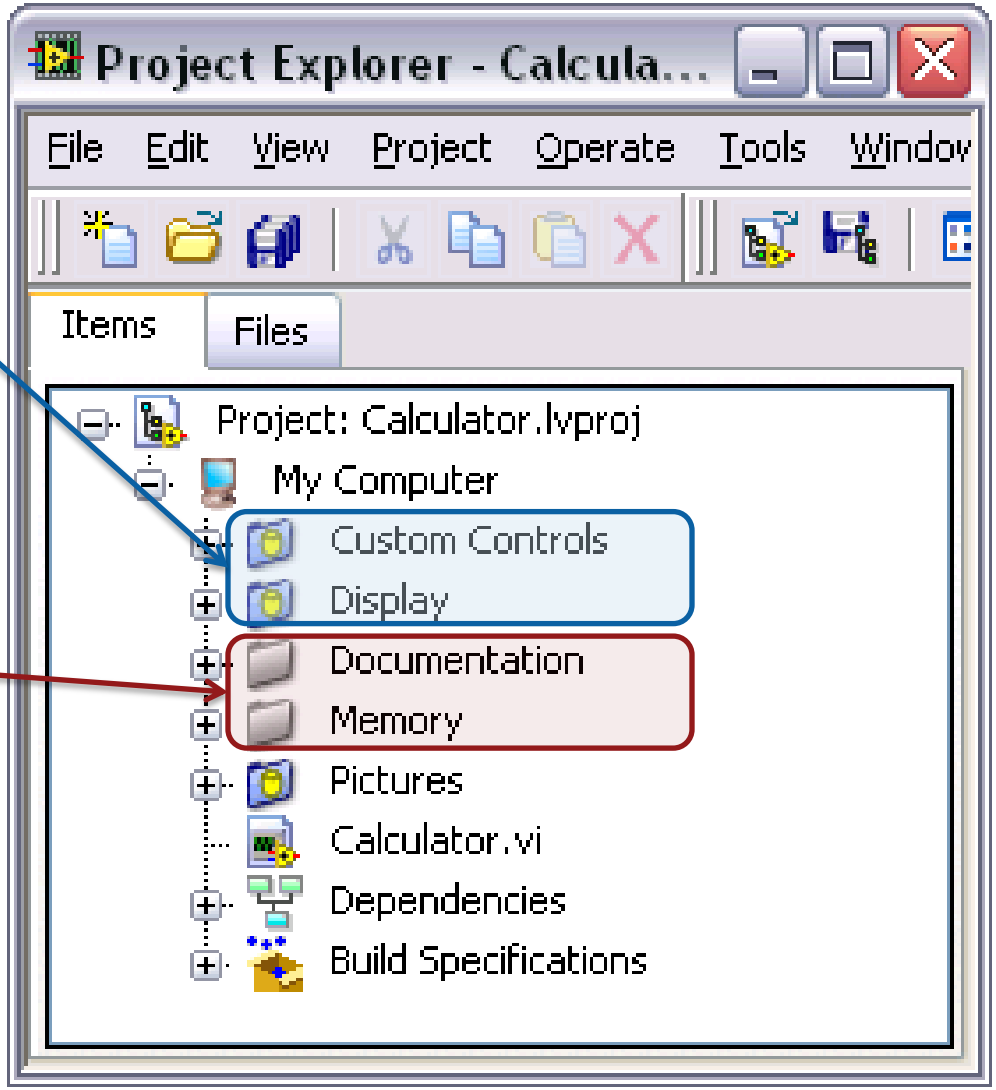
Managing Project Files

Auto-populating Folders

Update to reflect the contents of folders on disk

Virtual Folders

Customize how and where files are displayed

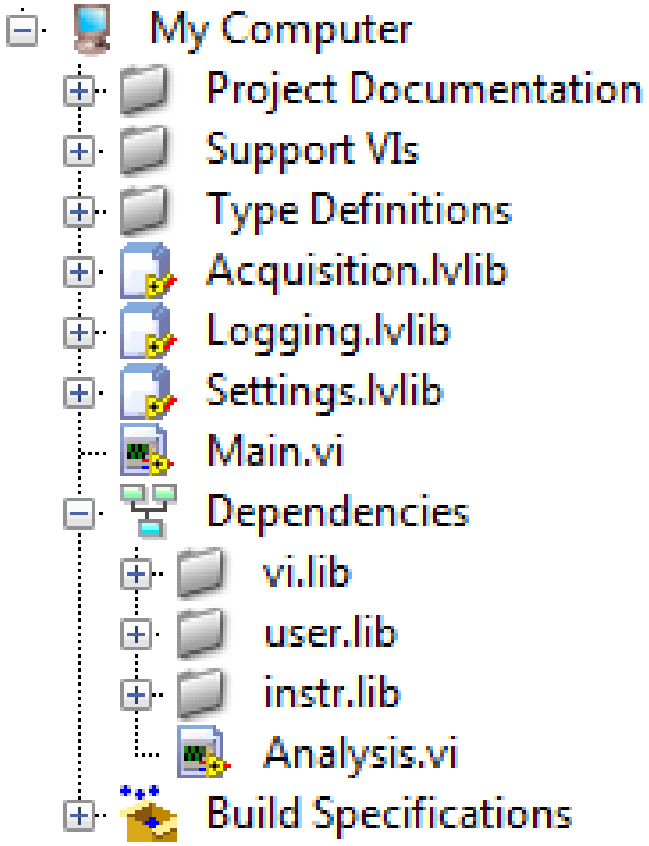


NATIONAL INSTRUMENTS

Project Dependencies

LabVIEW automatically identifies the files required by each item in a project.

- Ensure you're using the correct version of a subVI
- Understand which files should be added to the project



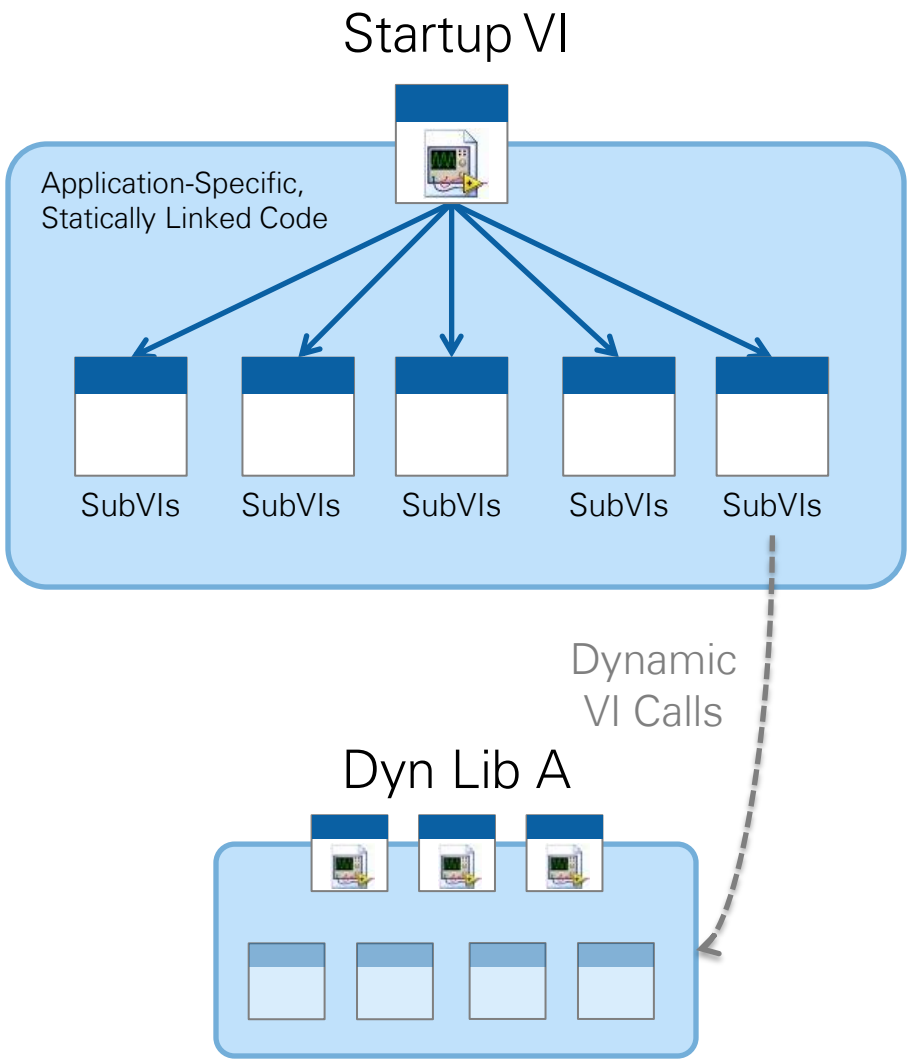
LabVIEW Directory Structure

vi.lib	Contains libraries of built-in VIs, which LabVIEW displays in related groups on the functions palette.
user.lib	Directory in which you can save controls and VIs you create. LabVIEW displays controls on User Controls palettes and VIs on User Libraries palettes.
instr.lib	Contains any installed instrument drivers. These drivers will appear on the Instrument I/O palette.

Files on Disk and the LabVIEW Project

DEMO

Dynamically Loaded Files



Dynamically loaded VIs are not in memory until they are loaded by the caller VI.

- Reduce load time of large caller VIs
- Optimize memory use

Will not be listed in Project Dependencies

Tracking Dynamically Loaded Files

Dynamically loaded files are not statically linked by any callers in the project. Any action that changes the path to a dynamically loaded file can prevent the project from loading the file.

To ensure that dynamically loaded files are in the correct location:

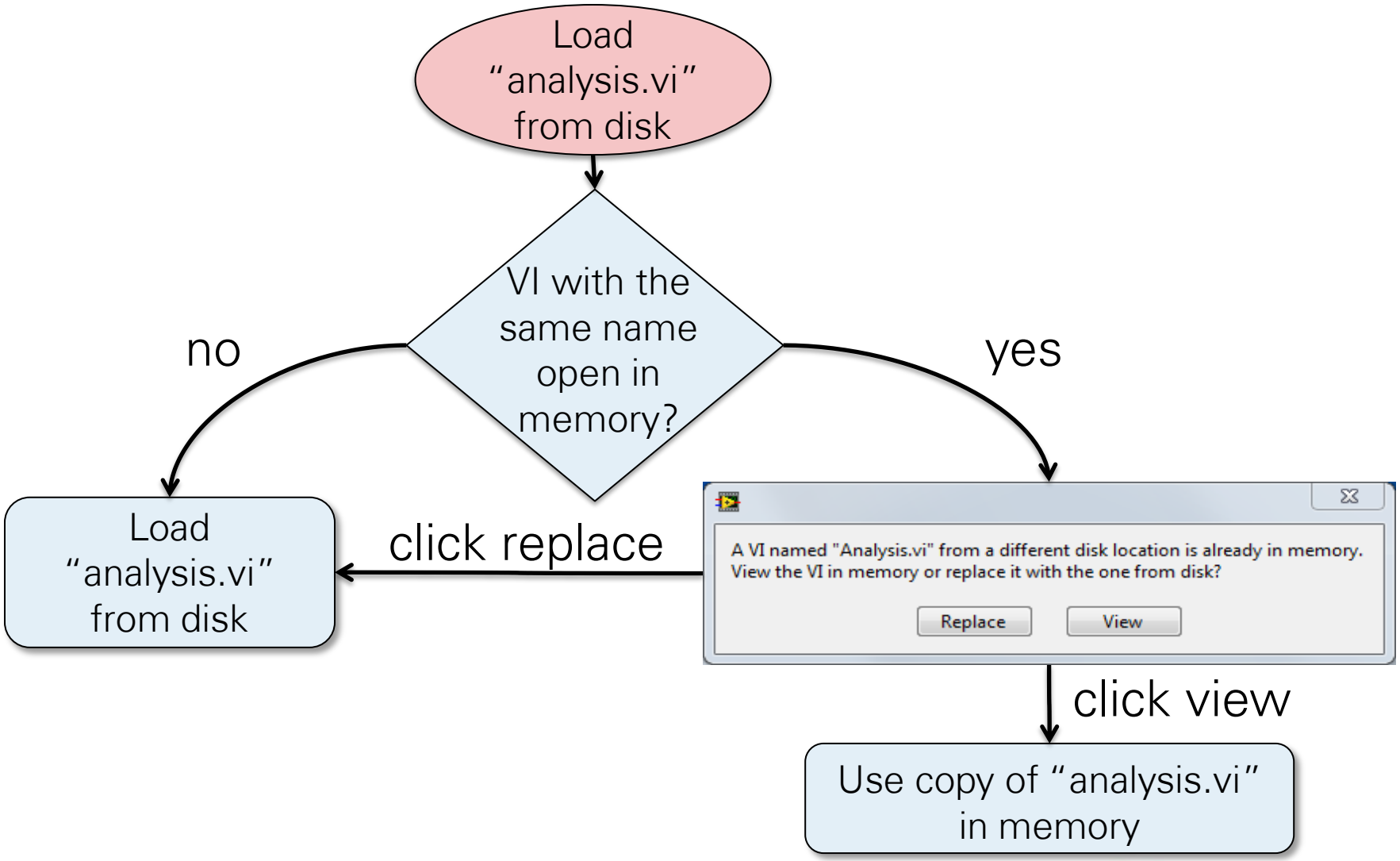
- Group the files in a separate folder.
- Refer to the files using relative paths.
- If you need to move a project or distribute an application, remember to include the folder in which you grouped dynamic dependencies.

LabVIEW Search Order

You cannot load two VIs with the same name in memory
Multiple VIs with the same name can exist on disk.

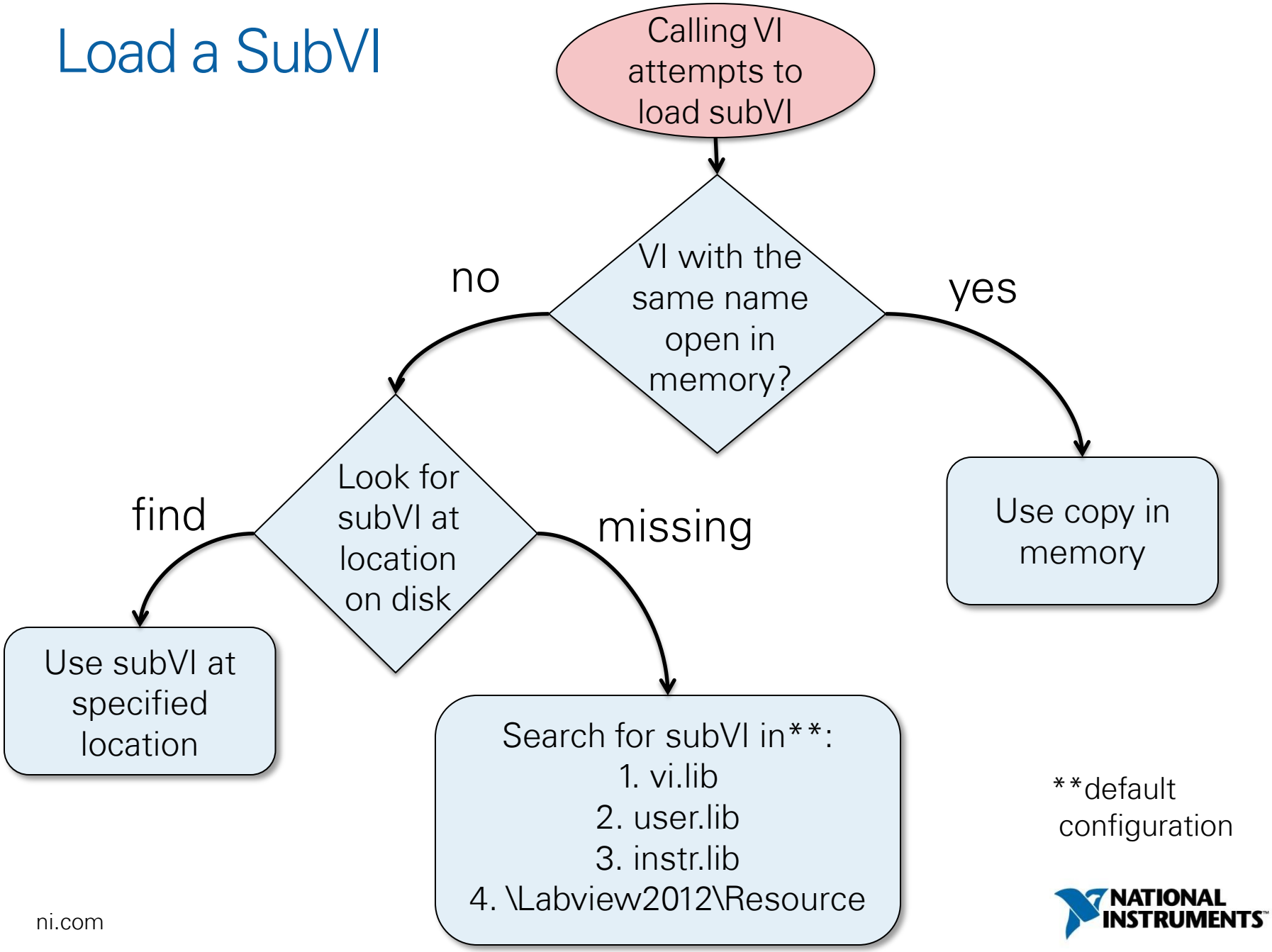
- What happens when you load a VI from disk
- What about when a calling VI loads a subVI?

Load a VI From Disk



NATIONAL INSTRUMENTS

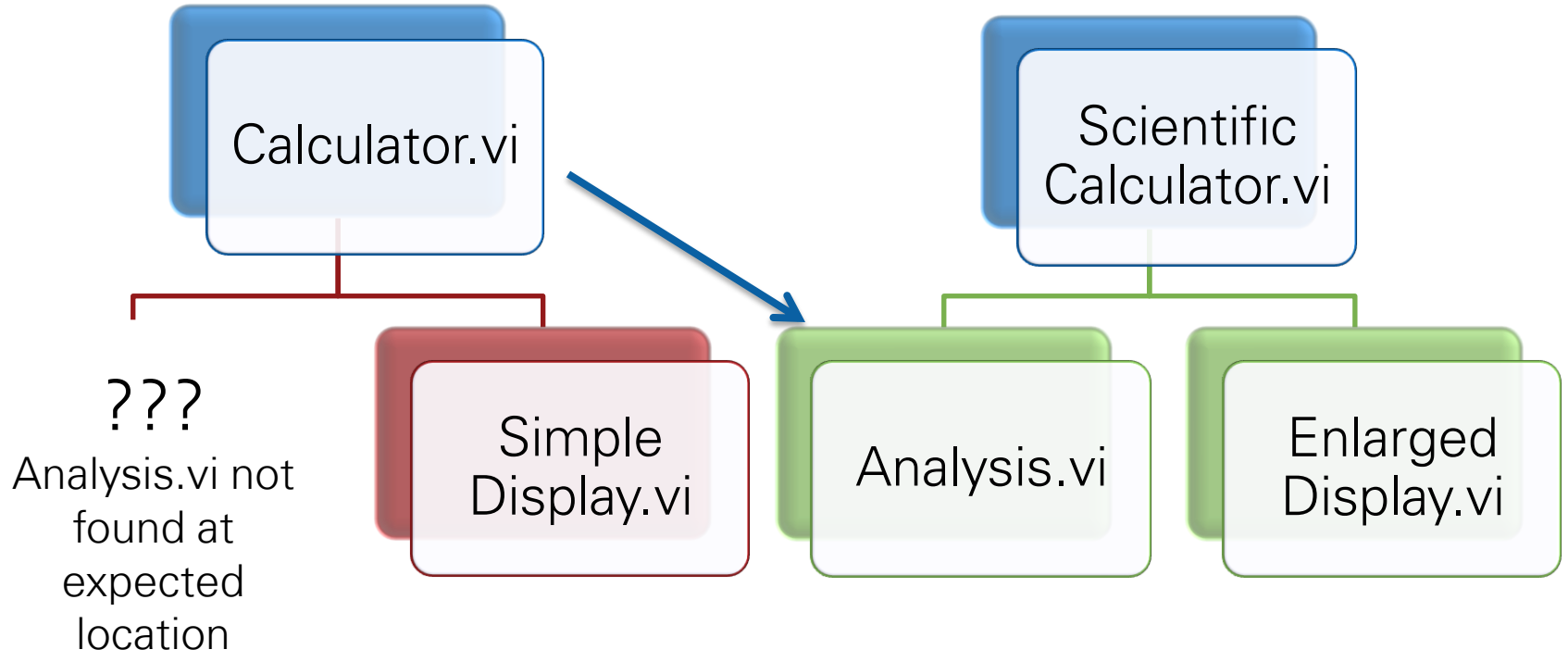
Load a SubVI



**default configuration

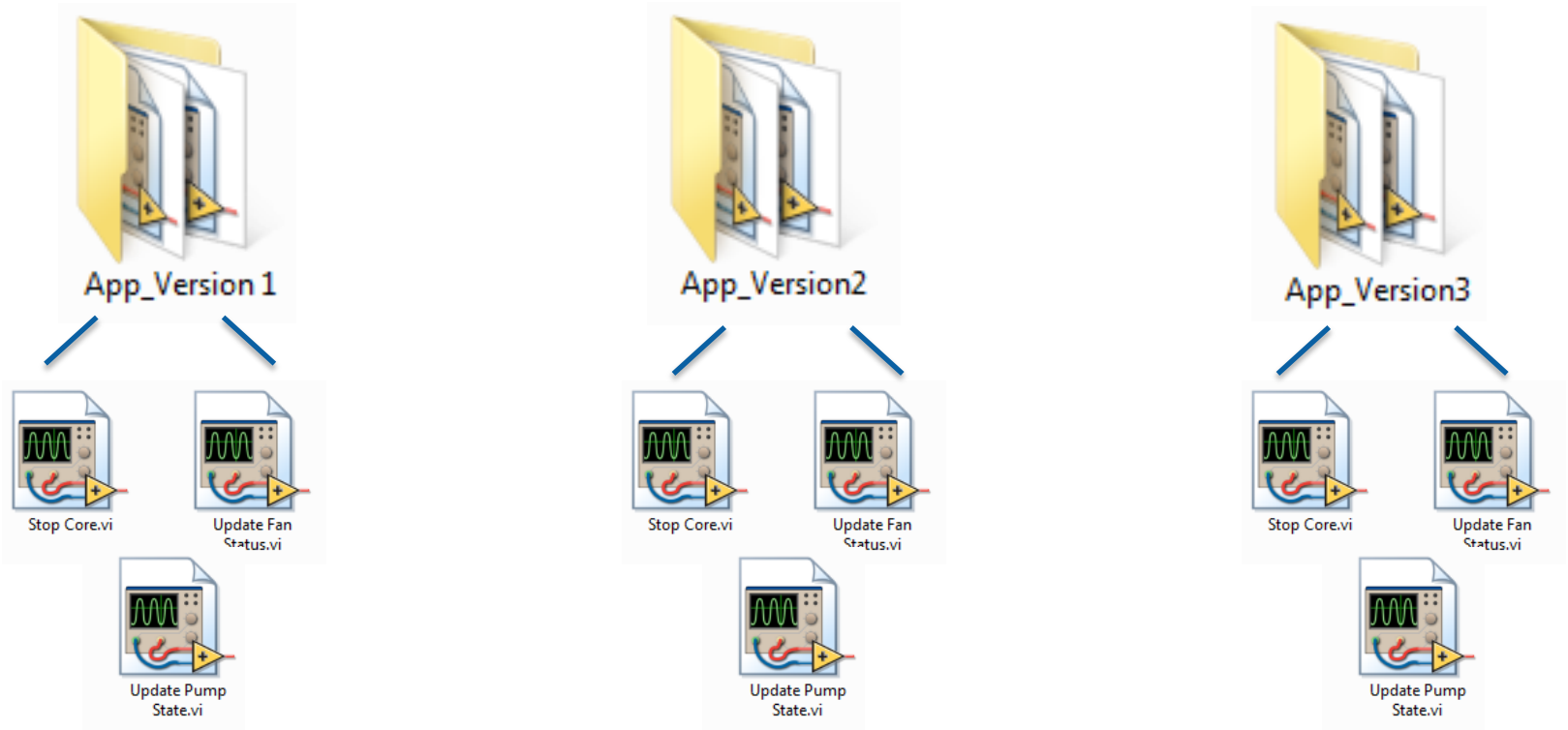
NATIONAL INSTRUMENTS

Cross-Linking Defined



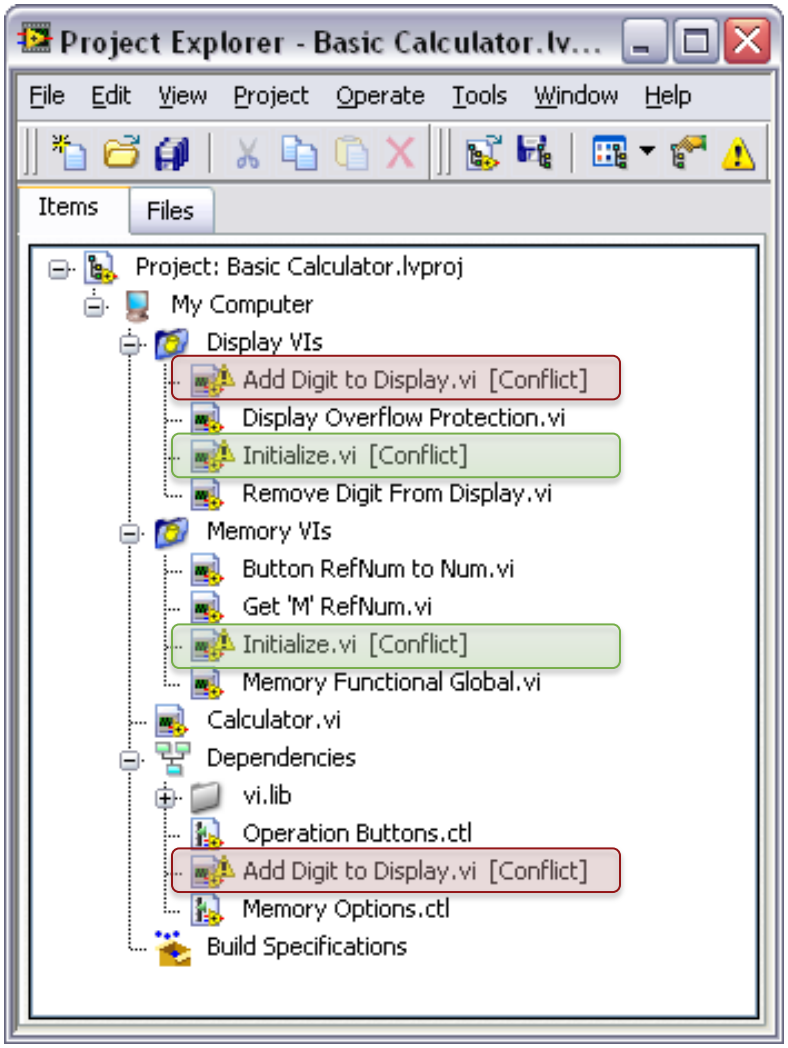
When Can Cross-Linking Occur?

If you back-up working directories by creating multiple copies, you will end up with many copies of a given VI on your machine.



NATIONAL INSTRUMENTS

Cross-Linking Notification



NATIONAL INSTRUMENTS

LabVIEW Search Order and Cross-Linking

DEMO

Preventing Cross-Linking

- ✓ Add all files to a LabVIEW project
- ✓ Be aware of dependencies
- ✓ Avoid duplicating code to create multiple back-ups
- ✓ Share code between projects via reuse library
- ✓ Ensure unique VI names

Summary

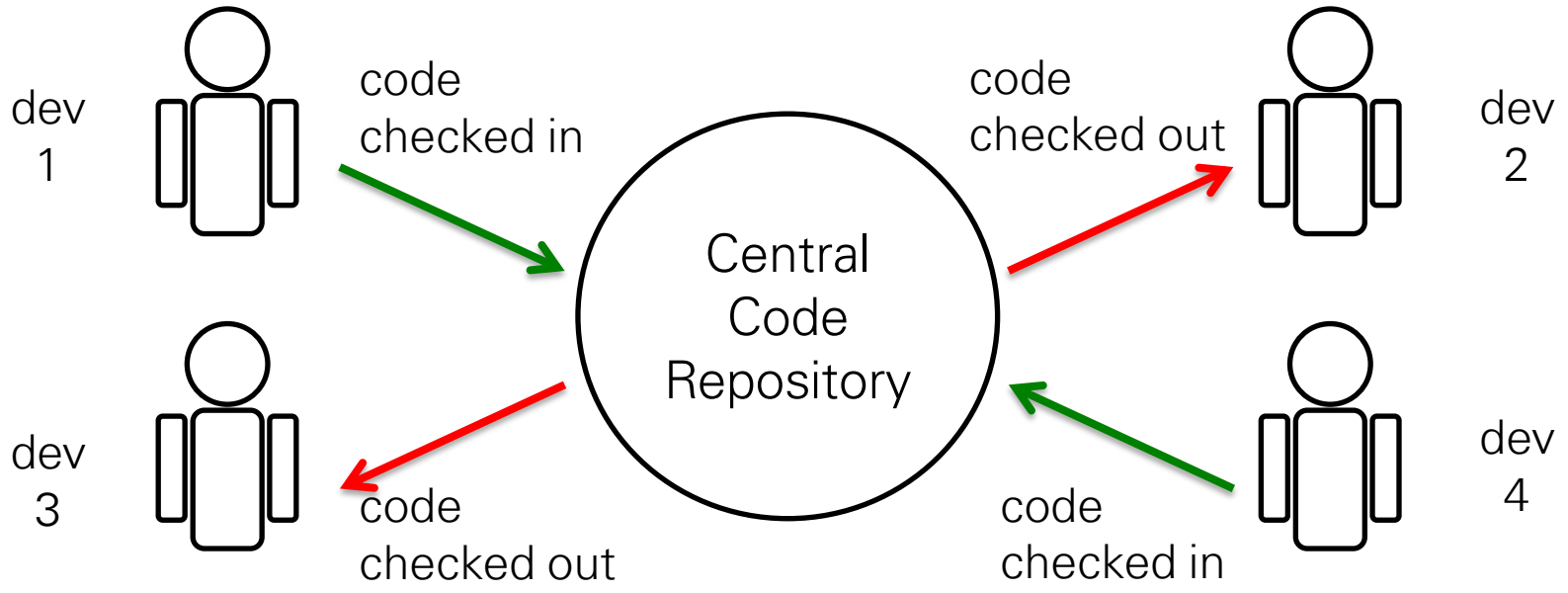
The project explorer is a valuable tool to organize applications and prevent common development pitfalls.

- Use Auto-populating and Virtual folders to customize organization
- Preserve linking by moving files with File View
- Group dynamically linked resources together
- Be aware of dependencies

Managing a Code Base with Source Code Control

Introducing Source Code Control

Source Code Control is a tool used to track, store, and manage all files related to an application during development.



NATIONAL INSTRUMENTS

Why Use Source Code Control?

Increase productivity by allowing multiple developers to contribute in a controlled environment

- Avoid code loss from overwriting

Manage files throughout the development cycle

- Code revision history let's developers track bugs quickly & document changes

Speed development with merge and diff tools

Source Code Control is a best practice for **any** modern software development project, regardless of complexity or team size.

What Tools Are Available?

Recommended

Perforce
Subversion

Additional Options

Microsoft Visual Source Safe
Microsoft Team Foundation Server
Rational ClearCase
PCVS (Serena) Version Manager
MKS Source Integrity
Seapine Surround SCM
Borland StarTeam
Telelogic Synergy

What Files Should I Put Under SCC?

- ✓ VIs
- ✓ Documentation
  - Track revisions to a Requirements Document with SCC
- ✓ Configuration Files
- ✓ Type Definitions

What about the *.lvproj file itself?

Should I Put My *.lvproj File Under SCC Too?

A LabVIEW *.lvproj file is an XML file that contains:

- Links to files contained in the project
- Settings for the project
- "Virtual items" such as build specifications

It's critical that all developers have the most recent version of the *.lvproj file to ensure they have all of the latest dependencies and resources

Should I Put My *.lvproj File Under SCC Too?

Anytime a file in the project is renamed or added, the *.lvproj file is altered and must be checked out of source code control, impacting all developers using the project

```
<Item Name="Acquisition.lvlib" Type="Library" URL="../Acquisition/Acquisition.lvlib"/>
<Item Name="Logging.lvlib" Type="Library" URL="../Logging/Logging.lvlib"/>
<Item Name="Settings.lvlib" Type="Library" URL="../Settings/Settings.lvlib"/>
<Item Name="My Analysis.vi" Type="VI" URL="../Analysis/My Analysis.vi"/>
<Item Name="Main.vi" Type="VI" URL="../Main.vi"/>
```

.lvlib files are represented in the Project File by library name only. As long as the name of the library remains the same, the contents of the library can change without modifying the .lvproj file.

Individual VIs are also represented by name in a project. This means that anytime a new VI is added or removed from the project, the .lvproj file is modified.

Best Practices for Managing Project Files with SCC

Determine your application framework before development begins.

- Create placeholders for all future code to avoid altering the project file
- Use .lvlib files to avoid modifying the project file

If a change needs to be made, have a single developer check out the project file and make the change

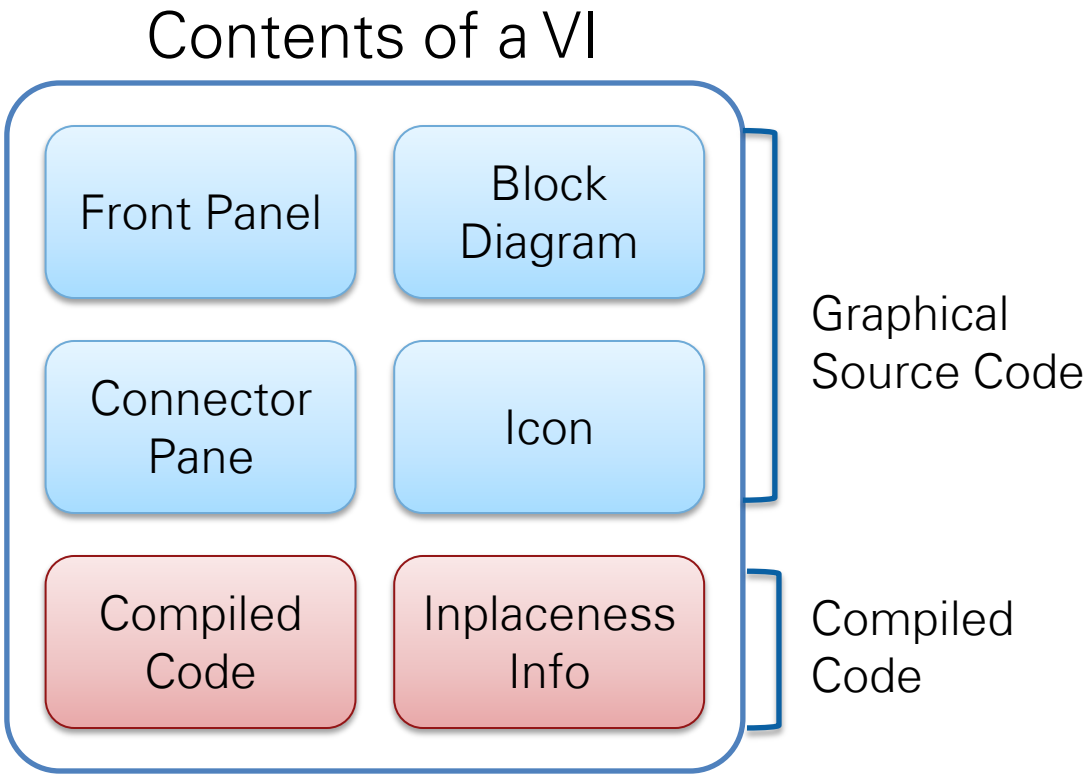
- Ensure that all other developers begin using the new version of the project file immediately

Configuring SVN Source Code Control with LabVIEW

DEMO

Considerations When Storing VIs Under SCC

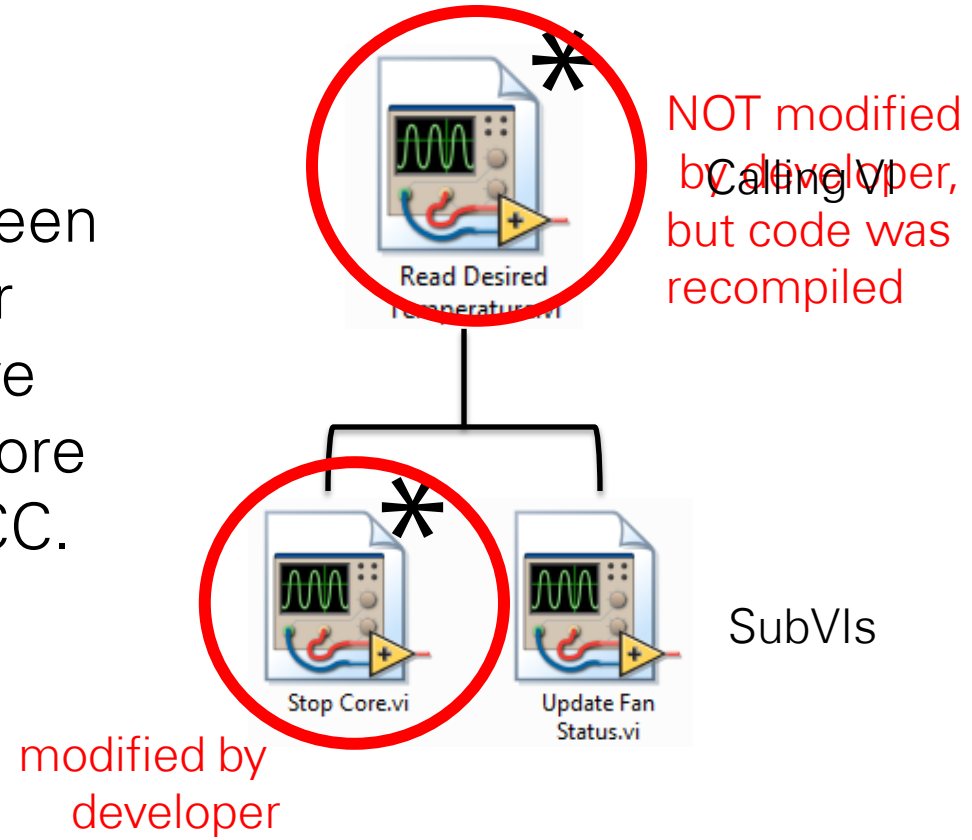
When you edit a VI, LabVIEW recompiles the VI code. **LabVIEW may also recompile the callers of that VI to optimize code.**



Considerations When Storing VIs Under SCC

When you edit a VI, LabVIEW recompiles the VI code. **LabVIEW may also recompile the callers of that VI to optimize code.**

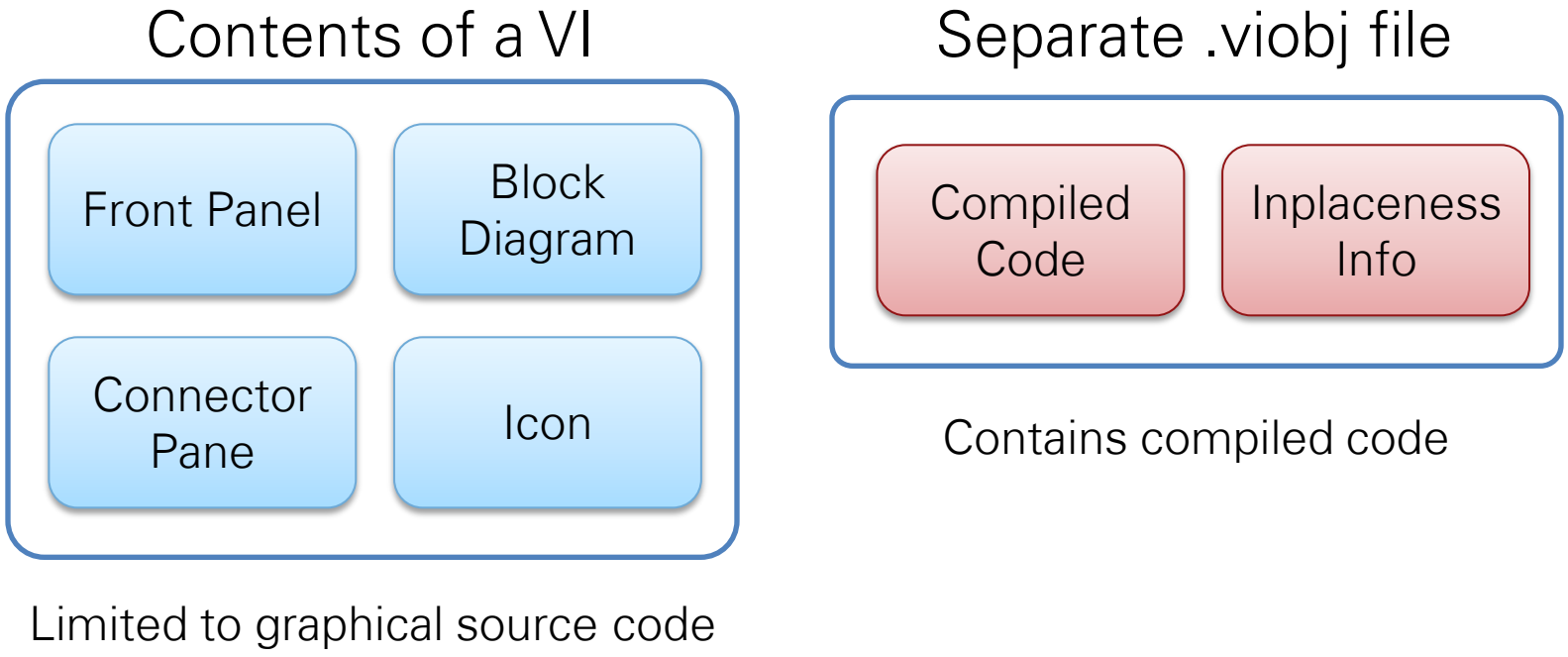
Calling VIs that have **not** been modified by a programmer may indicate that they have been modified, and therefore require resubmitting to SCC.



NATIONAL INSTRUMENTS

Considerations When Storing VIs Under SCC

Eliminate the need to re-save and re-submit files to source code control unless the graphical source code has been changed by the developer



Deciding When to Separate Compiled Code from VIs

Consider separating compiled code to:

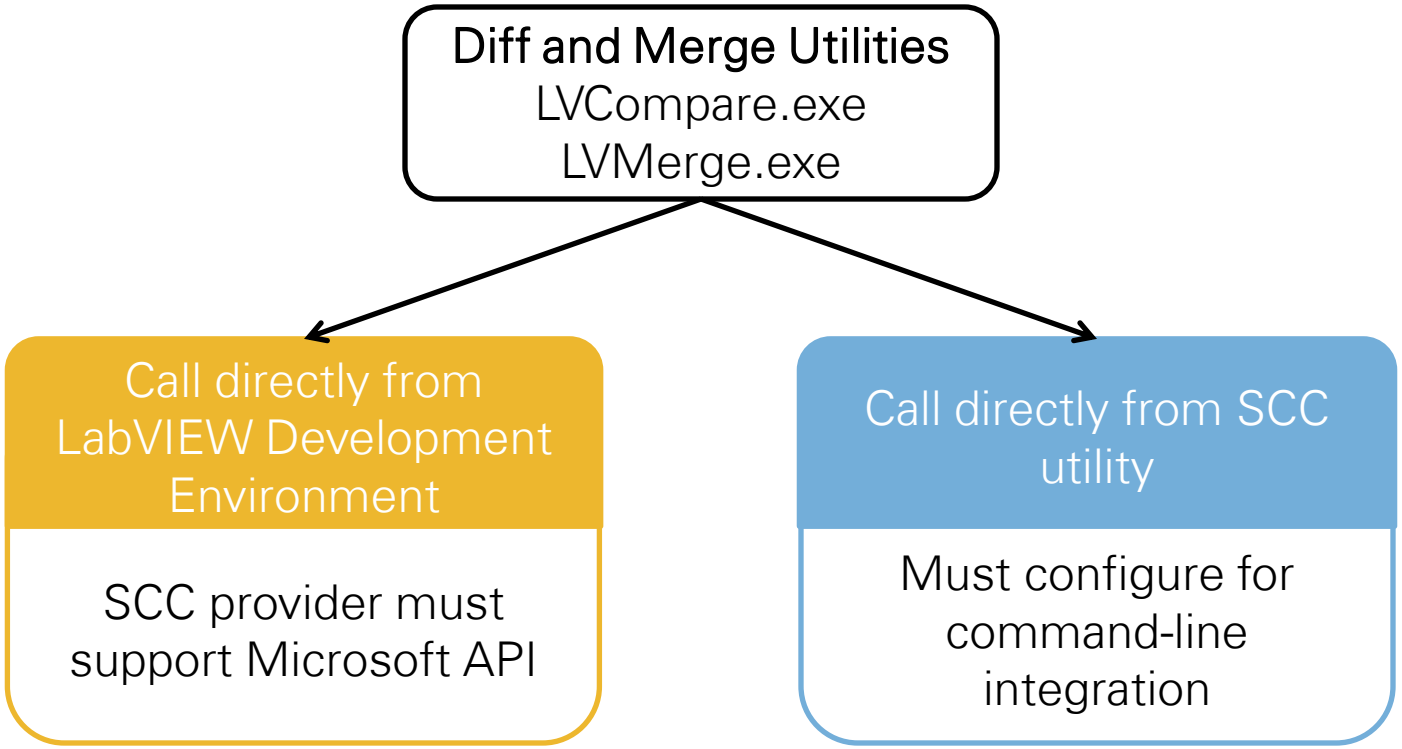
- Simplify source code control
- Prepare VIs under SCC to be upgraded to a new version of LabVIEW
- Improve load time of VIs

Do not separate compiled code:

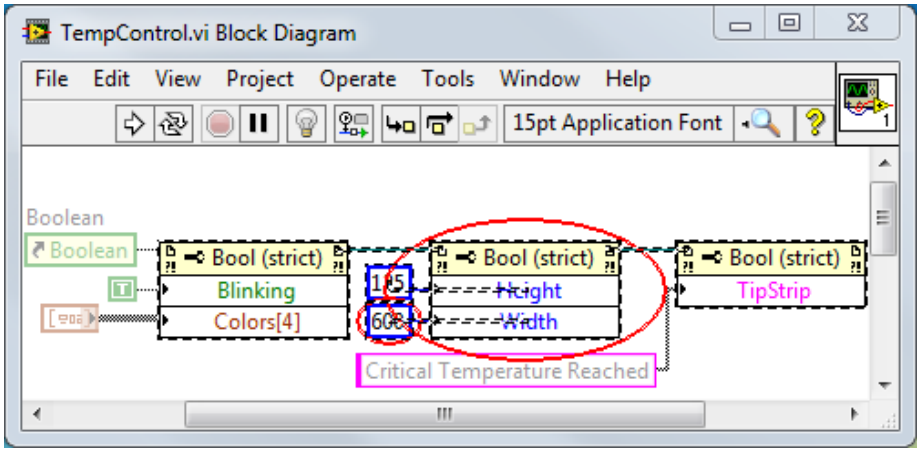
- If you intend to run the VI using the LabVIEW Run-Time Engine. In this case, consider building a source distribution instead

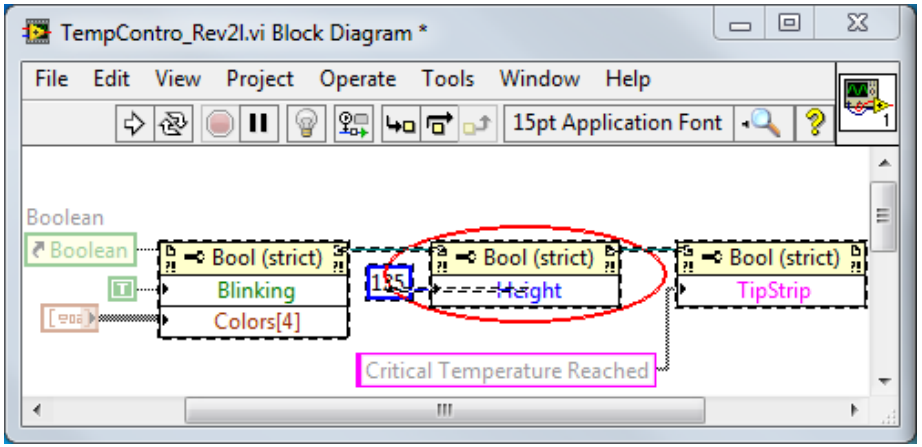
Graphical Diff and Merge Utilities

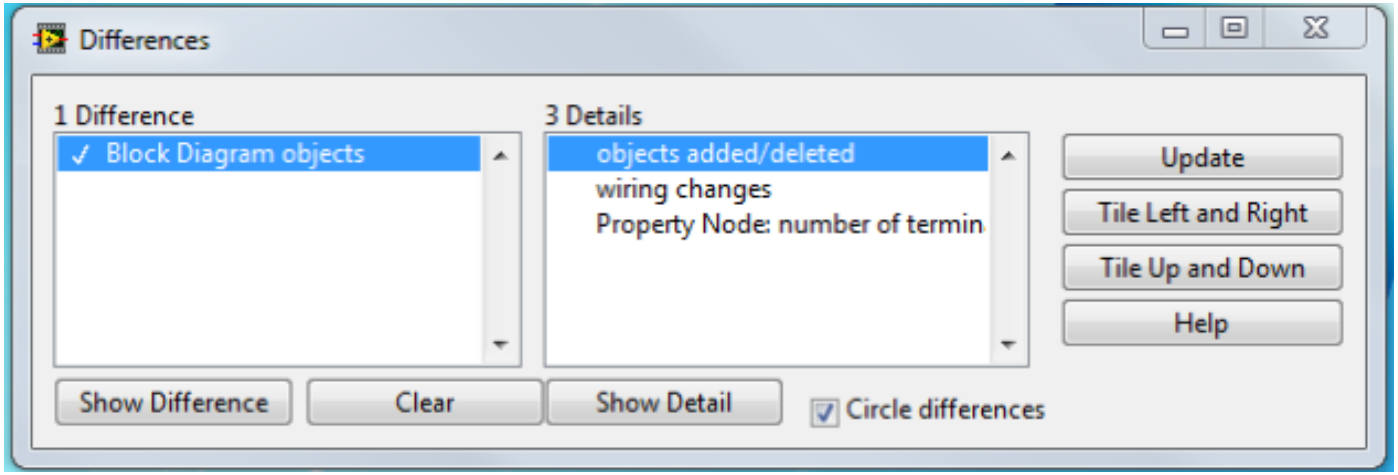
Since the source code for a VI is binary, specific comparison and merge utilities must be used.



Graphical Differencing







NATIONAL INSTRUMENTS

Graphical Merge

Simplify your development cycle by using Graphical Merge to automate code recombination.

Use this tool when modifications to a base VI are made by multiple individuals and saved separately.

Using Graphical Diff with SVN

DEMO

Group Development Recommendations

- **Use Source Code Control**
- Document changes at each submission
- Use VI Compare to assess changes
- Use VI Merge to reconcile code contributions

Building and Distributing Reuse Libraries

Challenge

How can I leverage common code among independent projects that all use different development cycles?

- What if the code I want to reuse is still being developed? How can I manage multiple revisions of reuse code?
- Copying common code is cumbersome and can lead to cross-linking

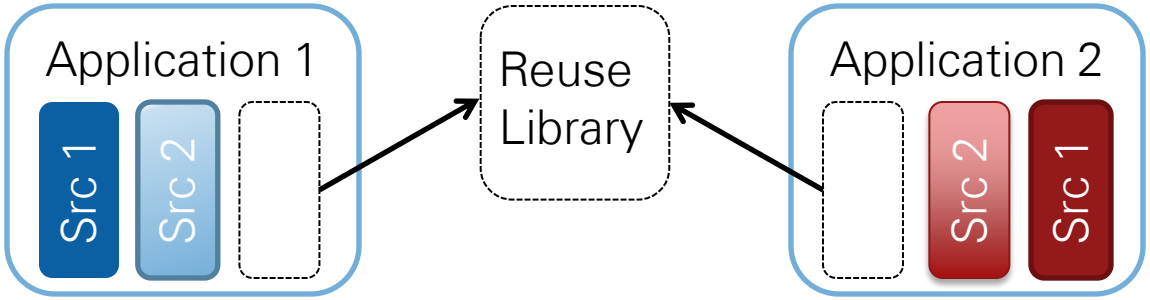
Solution: Build a reuse library

Managing Reuse Libraries

Reuse libraries are designed to be used across multiple projects. They may have a different development cadence than the projects they serve.

Reuse Libraries should be:

- De-coupled from the projects they are used in
- Easily upgraded or downgraded when maintaining multiple versions of an application



Source Distributions

A collection of files that you can package and send to users.

- Contains VI files, which allows you to move multiple VIs between developers as a single file

Create a directory or a zip file containing VIs

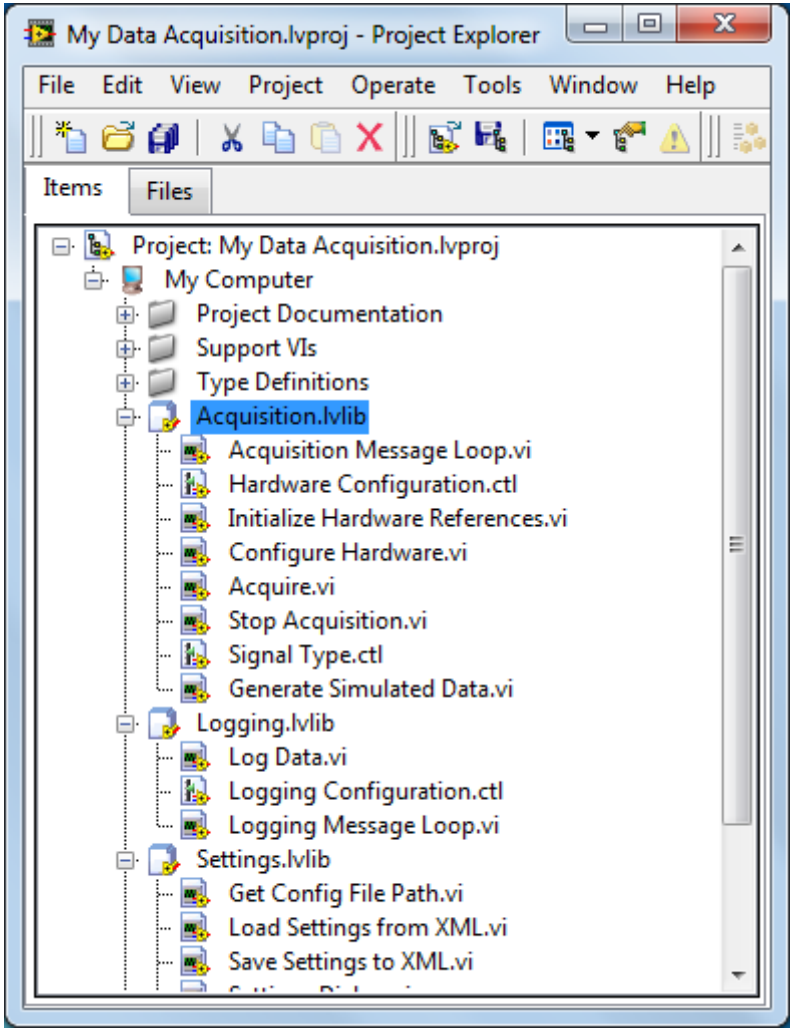
Configure which VIs are included, can choose to exclude vi.lib, user.lib, and instr.lib

The Project Library

A collection of VIs, typedefs, shared variables, palette files, and other files.

.lvlib file is an xml file that includes references to files that the library owns as well as library properties

A .lvlib file does not contain the actual files it owns



NATIONAL INSTRUMENTS

When Should I Use a Project Library?

Use a Project Library to:

- Encapsulate large sections of an application
- Organize a virtual hierarchy of items
- Qualify names of VIs to prevent cross-linking
- Modify contents without modifying the Project (*.lvproj) file
- Limit access to certain types of files by configuring the library to be public or private

Good choice when distributing an API

Packed Project Library Files *.lvlibp

A Packed Project Library is a precompiled .lvlib file that allows users to access public VIs in the library, but does not allow them to modify the code.

Why should I use a *.lvlibp file?

- Reduce build time for stand-alone applications
- Deploy fewer files by packaging multiple VIs into a single .lvlibp file
- Distribute an API of public VIs that cannot be modified

Using the Project Library

DEMO

Session Summary

Organizing and Managing LabVIEW Applications

Efficient Group Development Practices

Managing Reuse Libraries