Team Based Development with LabVIEW and Source Code Control

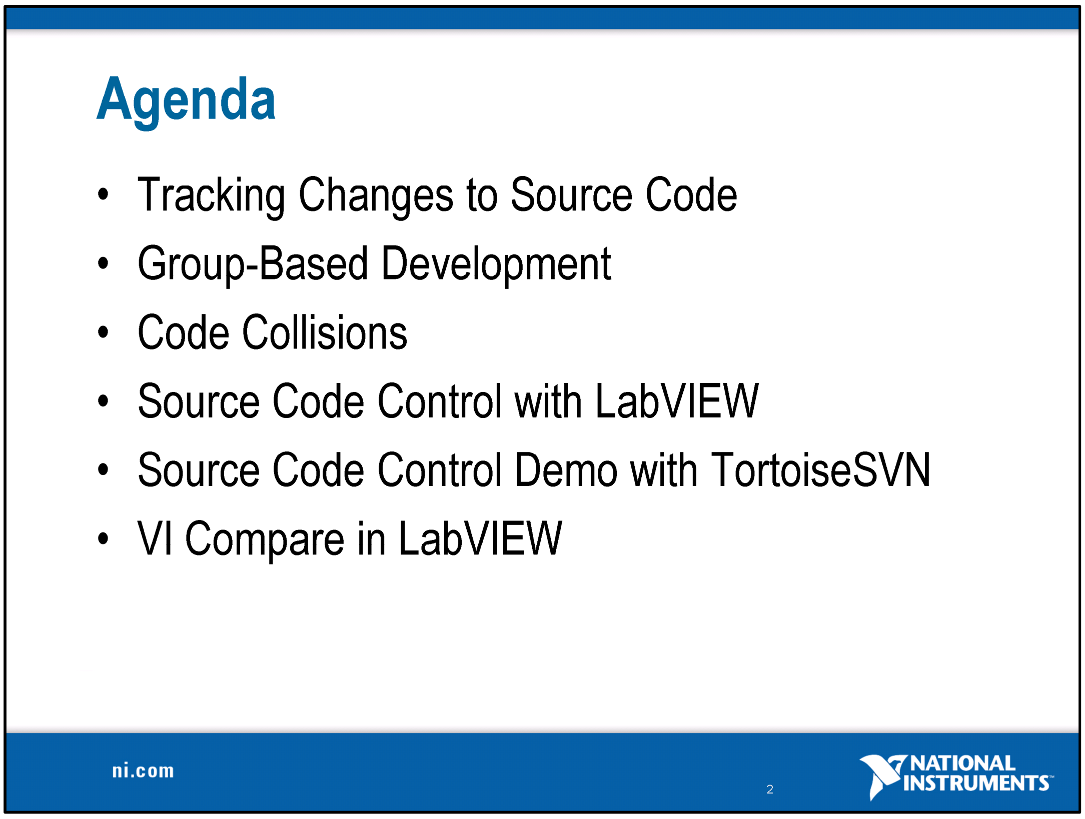
Mike Roberto – West LA District Sales Manager

mike.roberto@ni.com | (310) 372.5890

ni.com

NATIONAL INSTRUMENTS™

Presented by Mike Roberto to Aerospace Corp

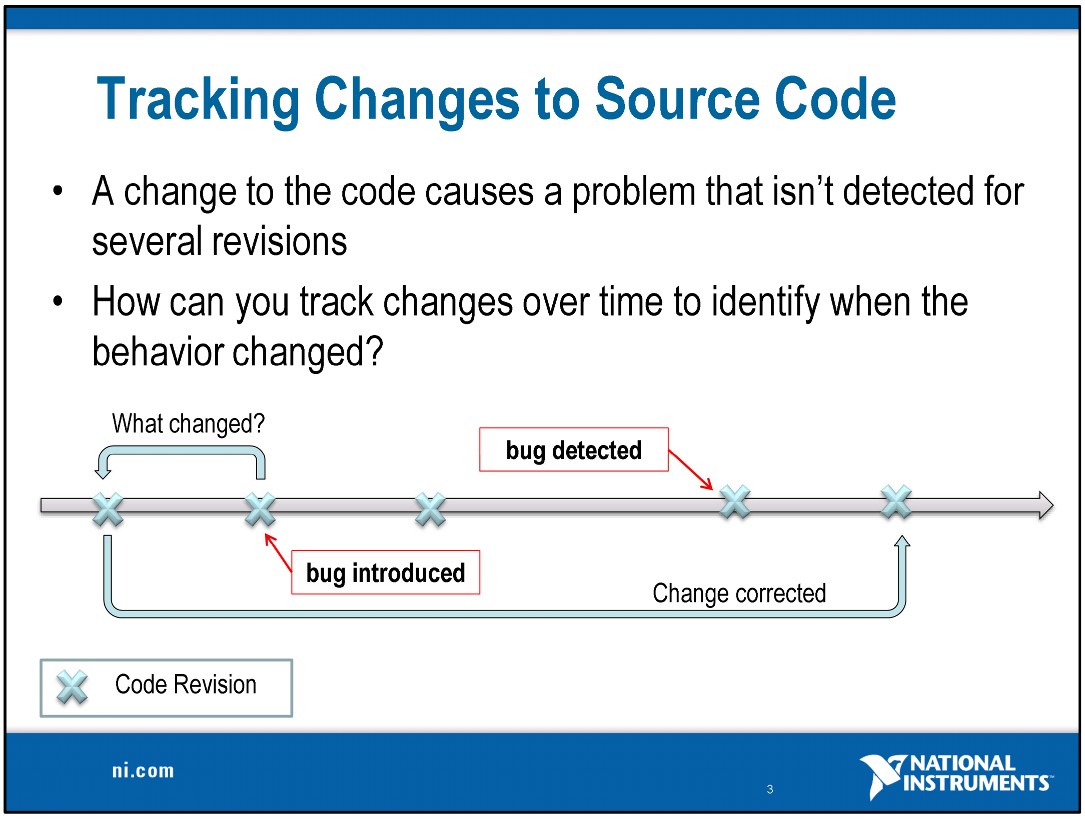


Agenda

- Tracking Changes to Source Code
- Group-Based Development
- Code Collisions
- Source Code Control with LabVIEW
- Source Code Control Demo with TortoiseSVN
- VI Compare in LabVIEW

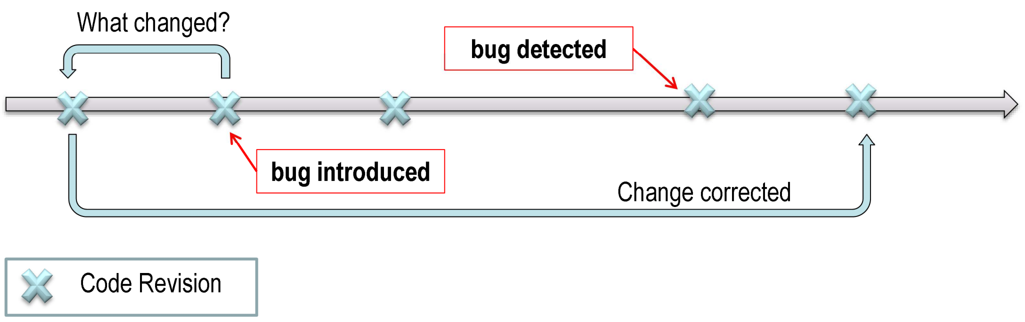
ni.com

2



Tracking Changes to Source Code

- A change to the code causes a problem that isn't detected for several revisions
- How can you track changes over time to identify when the behavior changed?



ni.com 3 NATIONAL INSTRUMENTS

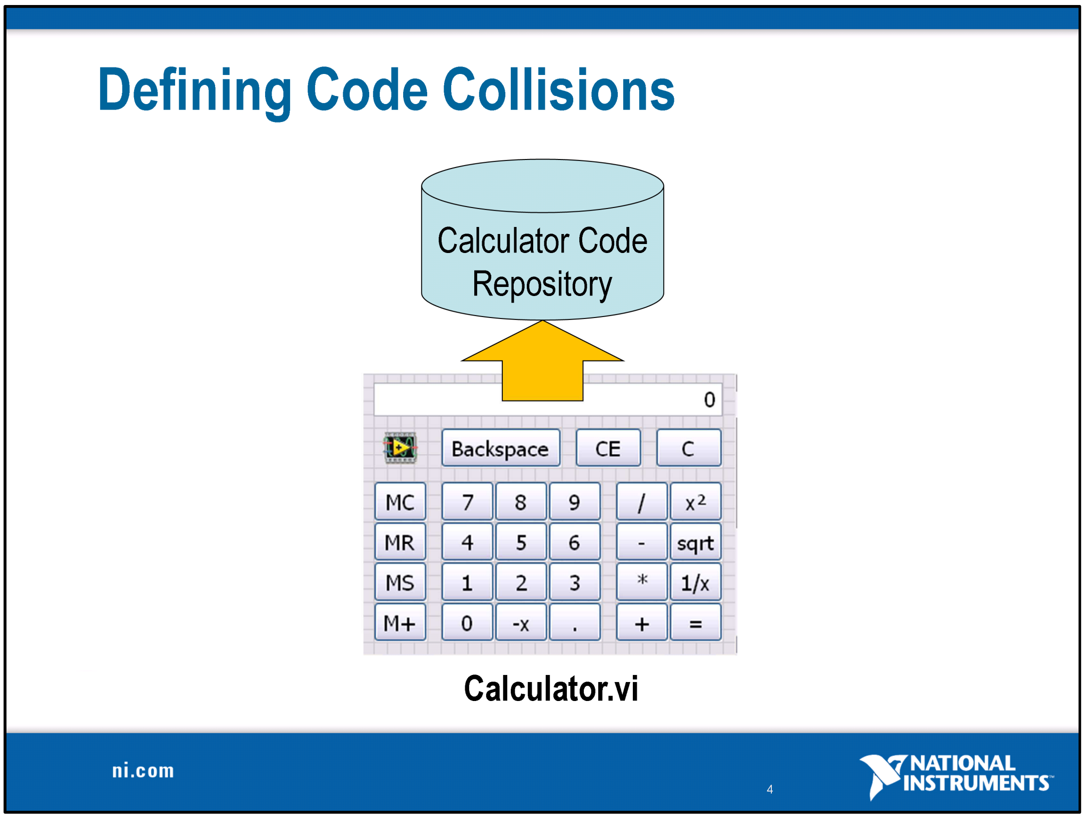
Lets examine difficult and sometimes frustrating problems that are common-place for software developers, but that can easily be improved or aided through the use of source code control.

When most people think of source code control, they typically think of group development and large, team-based efforts. The fact is that even single developers working on small projects can benefit greatly from the use of SCC for many reasons.

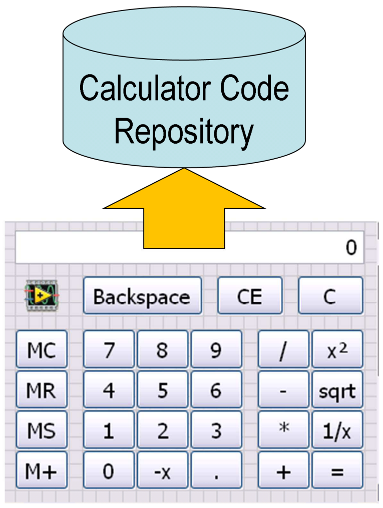
In the example shown above, consider an application where a problem that was introduced goes un-detected and un-checked through several iterations of the software. If it's a simple oversight it may be trivial to fix; however, for large projects, it can be difficult to isolate the source of a problem and may require stepping back through changes to see what caused it, when it was introduced, and who was responsible for the changes.

If you're developing without source code control, the only way to track older revisions of the code is to follow the 'stash-on-the-shelf' method. This can be accomplished a number of ways, either through zipping files up and renaming them, or storing them in different folder with a label that reflects which version they are. None of these methods are efficient or easy to follow, and almost all of these will lead to cross-links where you don't know which version of which dependency you're using. If you're not using source code control, it's more likely that you don't even have access to older versions and you're stuck trying to track down the problem on your own.

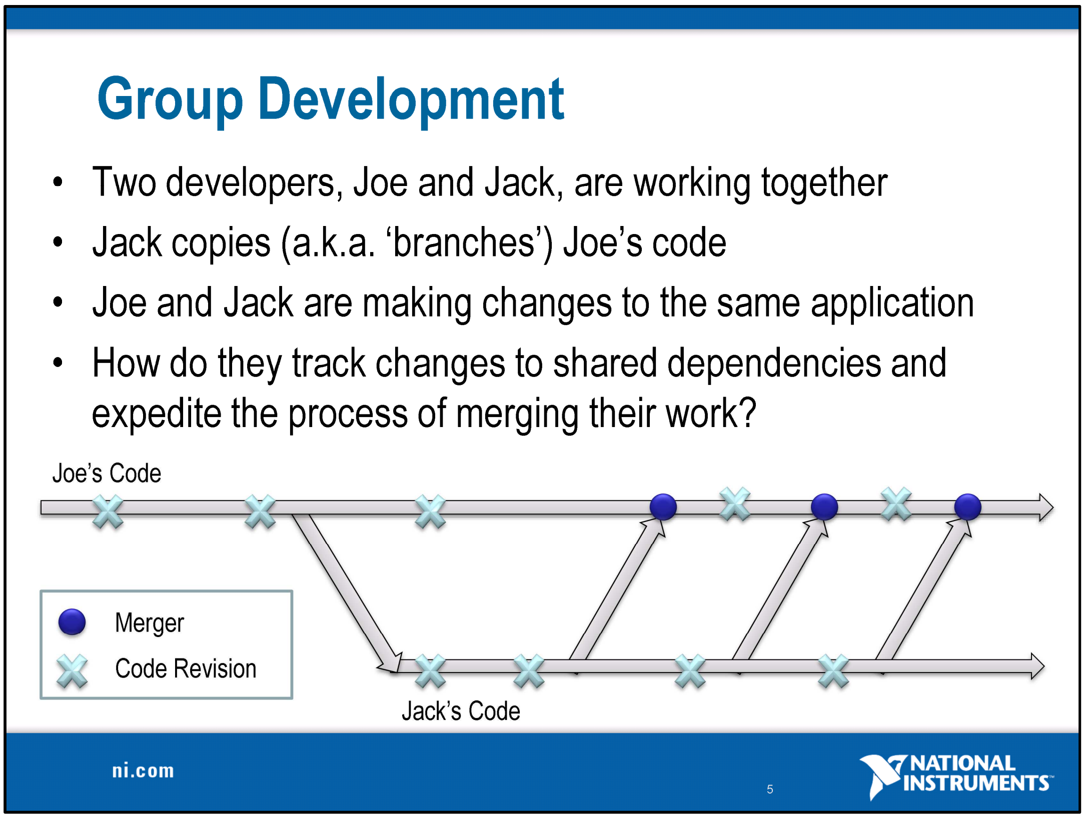
On the other hand, if you have source code control, you automatically have access to old versions of code and can either restore individual files or roll everything back to a prior version to examine behavior. Additionally, you can actually perform graphical diff to see exactly what has been changed between versions.



Defining Code Collisions

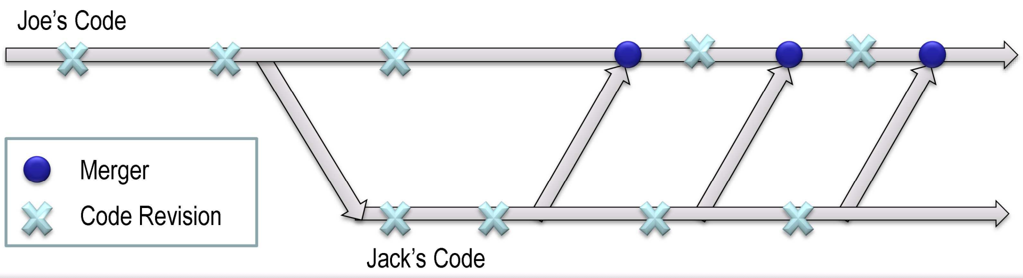


Calculator.vi



Group Development

- Two developers, Joe and Jack, are working together
- Jack copies (a.k.a. 'branches') Joe's code
- Joe and Jack are making changes to the same application
- How do they track changes to shared dependencies and expedite the process of merging their work?



ni.com 5 NATIONAL INSTRUMENTS

Another common challenge stems from team-based development. The illustration at the bottom is an example of the development process for a single application when two developers are working on it.

Joe is the original developer of the application, but at a certain point he apparently decides to enlist the help of a colleague, Jack, to work on another section of the same application. For many, you can probably relate to the practice of e-mailing a friend some code, or perhaps exchanging it across a shared network drive – this scenario is very similar.

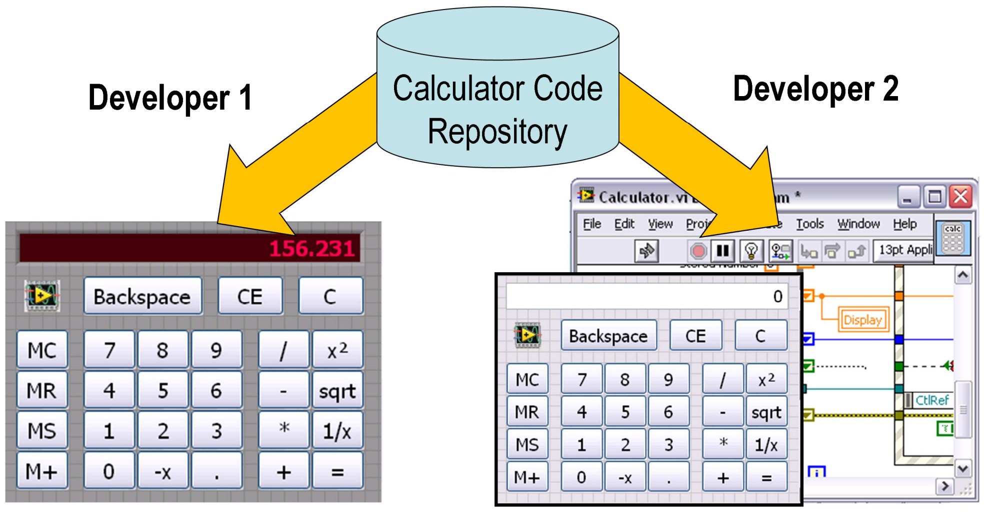
Note that the x's indicate where changes are being made. While Joe and Jack are working on separate parts of the application, it may become necessary for one or both of them to modify code that the other person is currently using.

If Joe is to make a change to a dependency Jack uses, moving his files back over could quickly lead to confusion, cause unexpected behavior, or cause problems that are subtle and therefore Jack is not immediately aware of.

If they are lucky enough to be aware of changes, the red circles indicate moments in time where they would have to invest in combining these changes. For a large body of code, this could be a long and arduous process. It also assumes that careful consideration was given to architecture, code interfaces, and the design of the code before attempting to have multiple people working in the source code.

While this may work for some, there is a lot of room for something to go wrong or unnoticed, and at the very least, Joe and Jack will need to be prepared to spend a significant amount of time manually combining changes – a process which is also error prone and can often cause more problems than it solves.

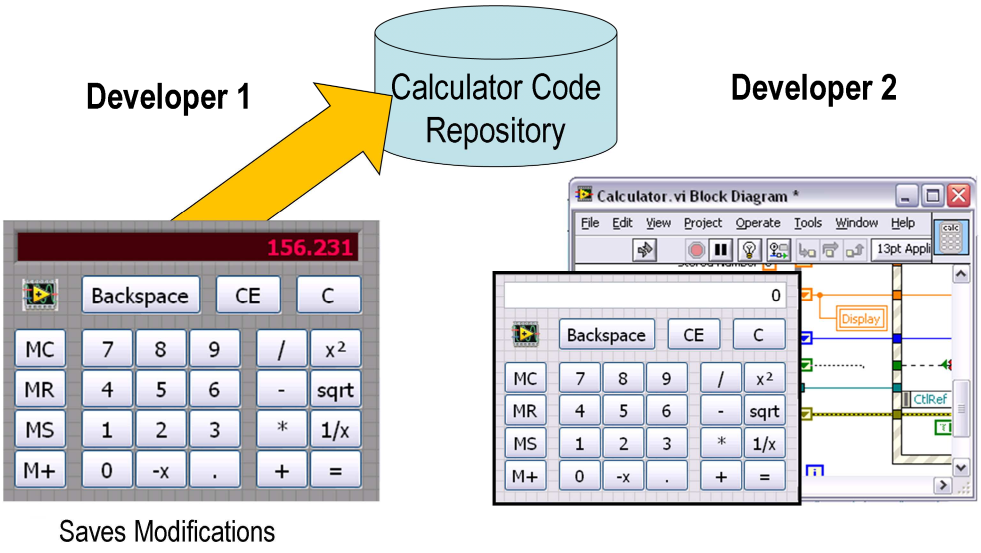
Defining Code Collisions



Changes Display Color

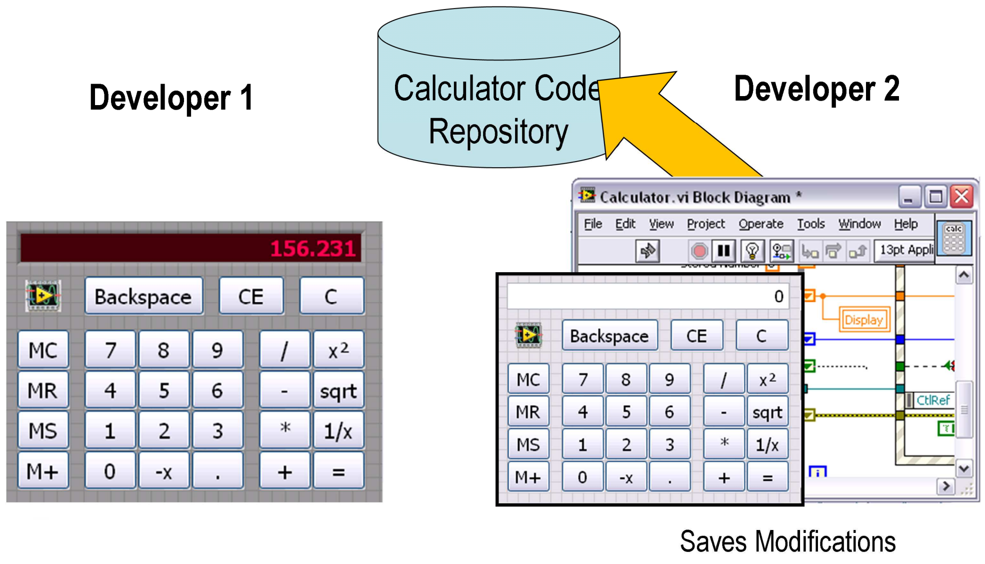
Changes Block Diagram Code

Defining Code Collisions

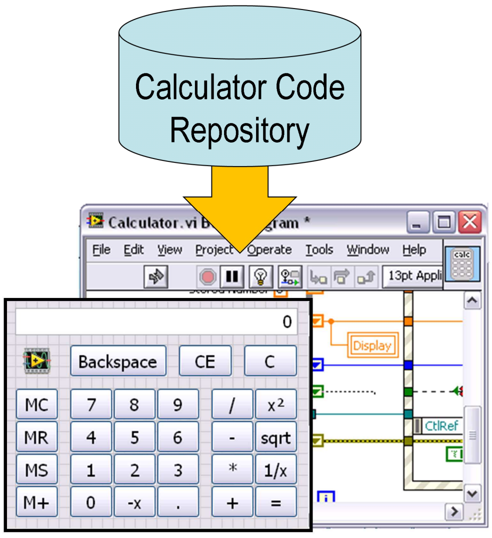


Developer 1
Calculator Code Repository
Developer 2
156.231
Backspace
CE
C
MC
MR
MS
M+
sqrt
1/x
Calculator.vi Block Diagram *
File Edit View Project Operate Tools Window Help
Display
CtlRef
Saves Modifications

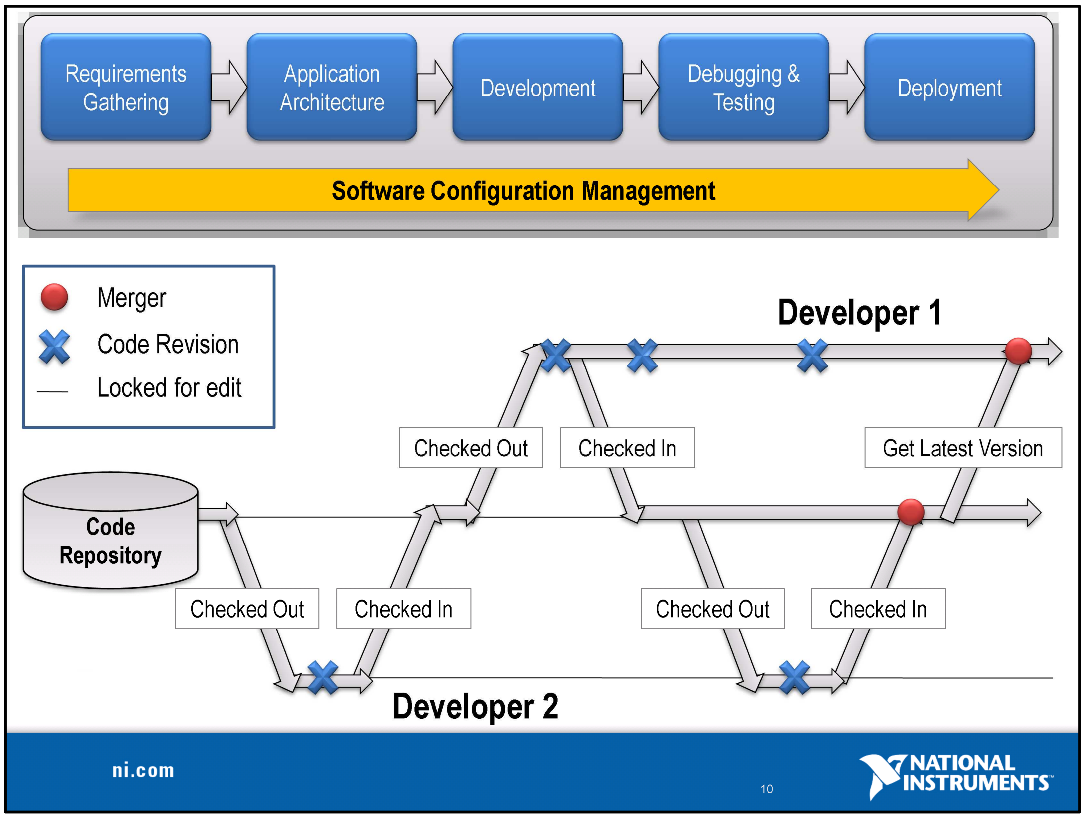
Defining Code Collisions


Developer 1
Calculator Code Repository
Developer 2
156.231
Backspace
CE
C
MC
MR
MS
M+
Calculator.vi Block Diagram *
File Edit View Project Operate Tools Window Help
Display
CtlRef
Saves Modifications

Defining Code Collisions


Calculator Code Repository
Calculator.vi Block Diagram *
File Edit View Project Operate Tools Window Help
13pt Appli
Display
CtlRef
0
Backspace CE C
MC 7 8 9 / x²
MR 4 5 6 - sqrt
MS 1 2 3 * 1/x
M+ 0 -x . + =

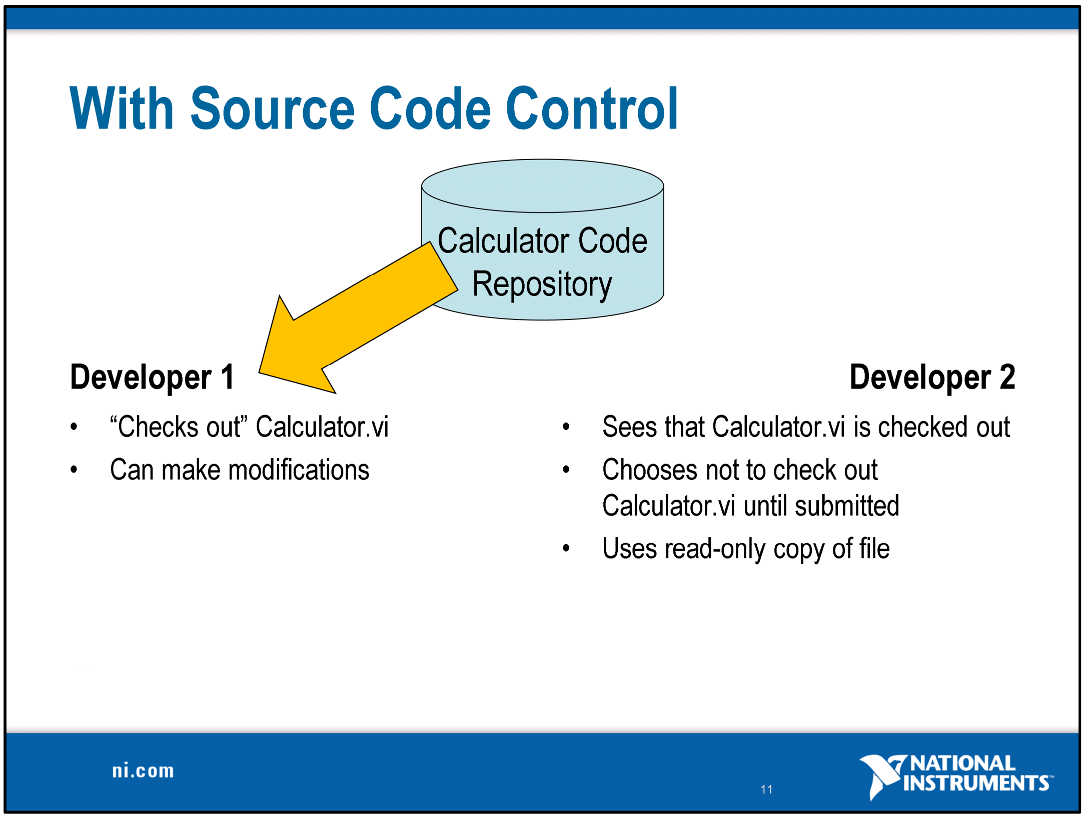
Developer 1 loses work because changes are over-written



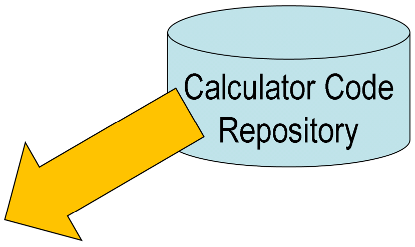
This is an illustration of how source code control improved the development process, in particular, for team-based projects. Instead of just copying code, developers now use source code control to check it out. This is effectively a developer's method for information a higher authority (in this case, the SCC provider) that they have an intent to modify the source code.

At this point, the behavior and limitations of the SCC interface are setup at the discretion of an administrator. Some may choose, for example, to lock the code when someone has indicated they choose to modify. Others may allow multiple users to have it checked out simultaneously.

If it becomes necessary to combine changes made by developers to the same piece of code, SCC can help with merging by allowing the two copies of the file to exist in the repository. Additionally, merge is available via the command line, so the automated LVMerge functionality can be invoked in the same manner as many text based merge applications.



With Source Code Control

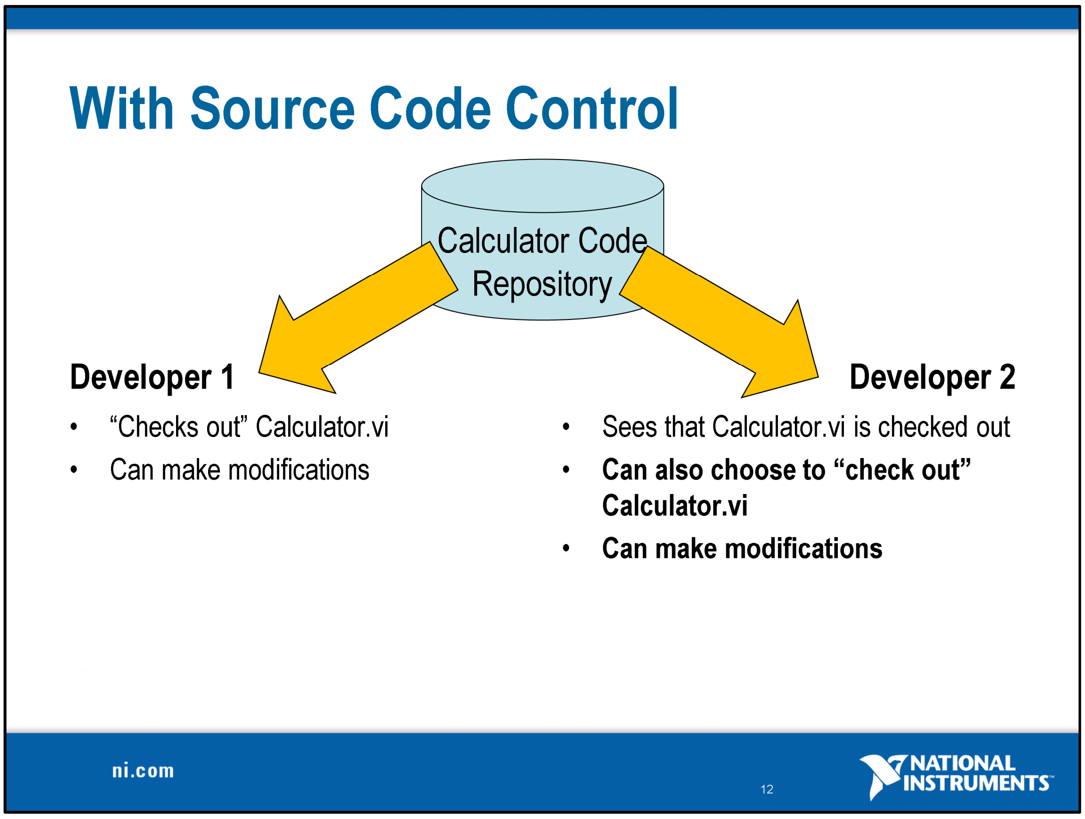


Developer 1

- "Checks out" Calculator.vi
- Can make modifications

Developer 2

- Sees that Calculator.vi is checked out
- Chooses not to check out Calculator.vi until submitted
- Uses read-only copy of file



With Source Code Control

Calculator Code Repository

Developer 1

- "Checks out" Calculator.vi
- Can make modifications

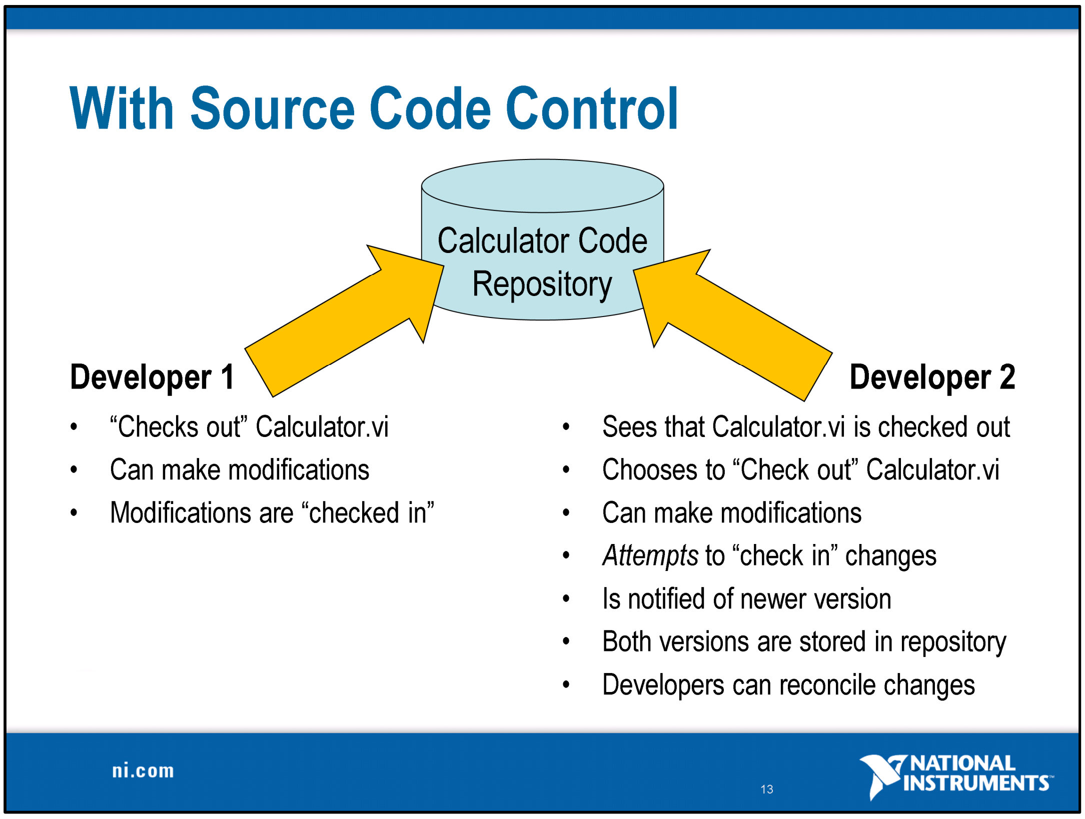
Developer 2

- Sees that Calculator.vi is checked out
- **Can also choose to "check out" Calculator.vi**
- **Can make modifications**

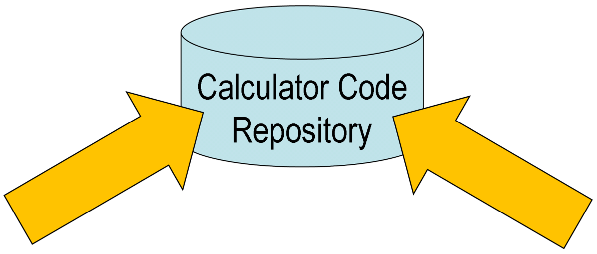
ni.com

12

NATIONAL INSTRUMENTS™



With Source Code Control



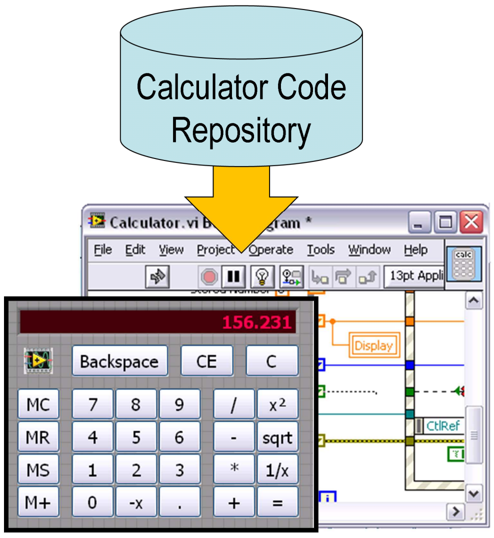
Developer 1

- “Checks out” Calculator.vi
- Can make modifications
- Modifications are “checked in”

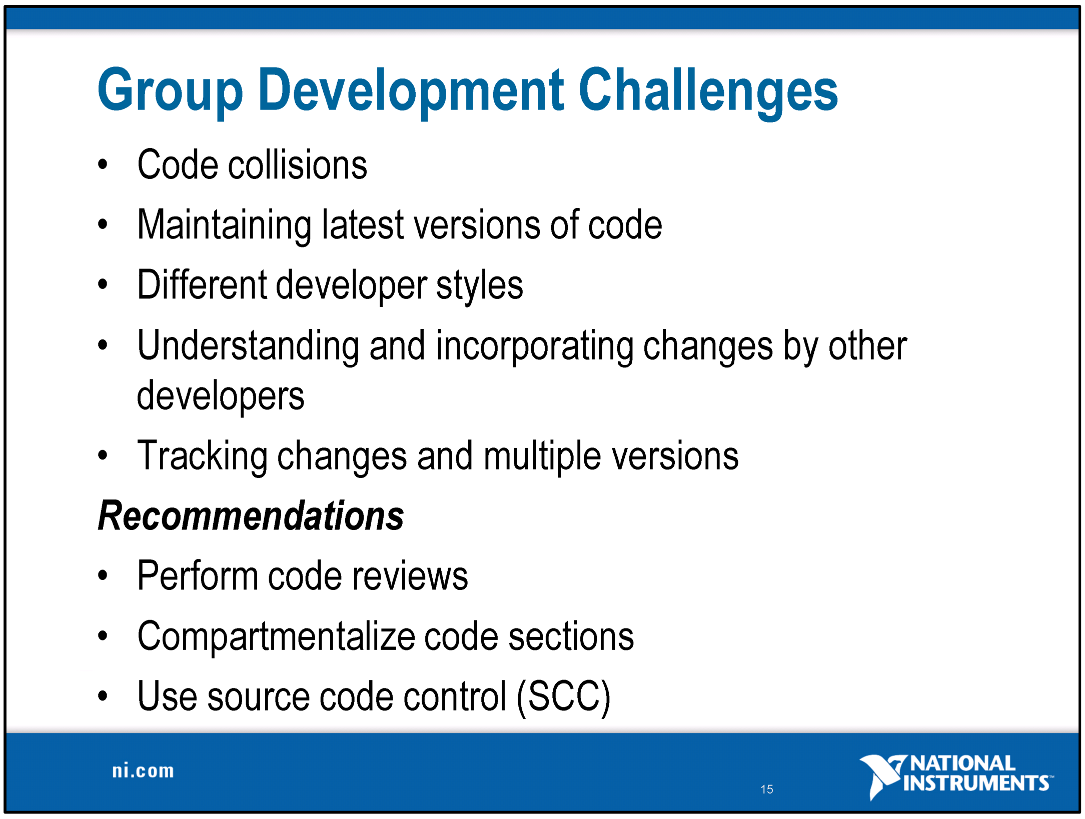
Developer 2

- Sees that Calculator.vi is checked out
- Chooses to “Check out” Calculator.vi
- Can make modifications
- *Attempts* to “check in” changes
- Is notified of newer version
- Both versions are stored in repository
- Developers can reconcile changes

With Source Code Control


Calculator Code Repository
Calculator.vi Block Diagram *
File Edit View Project Operate Tools Window Help
13pt Appli
Display
CtlRef
156.231
Backspace
CE
C
MC
MR
MS
M+
7 8 9 / x²
4 5 6 - sqrt
1 2 3 * 1/x
0 -x . + =

Final version has both developers modifications



Group Development Challenges

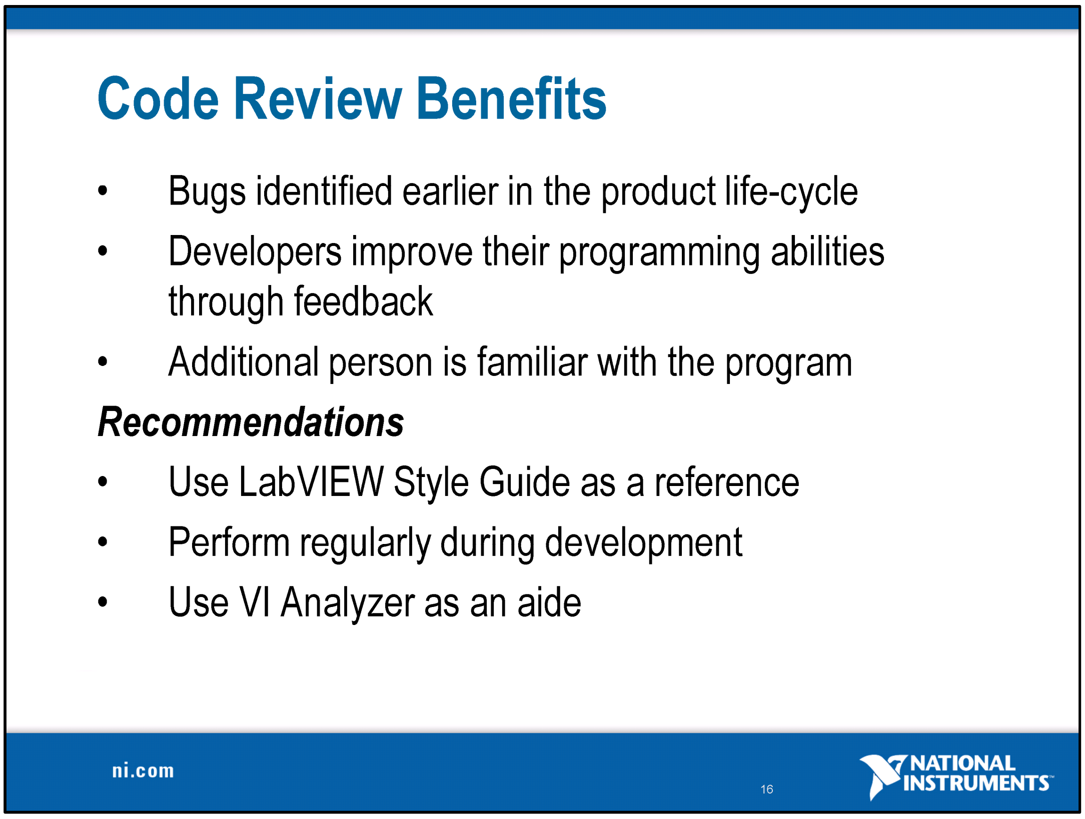
- Code collisions
- Maintaining latest versions of code
- Different developer styles
- Understanding and incorporating changes by other developers
- Tracking changes and multiple versions

Recommendations

- Perform code reviews
- Compartmentalize code sections
- Use source code control (SCC)

ni.com

15

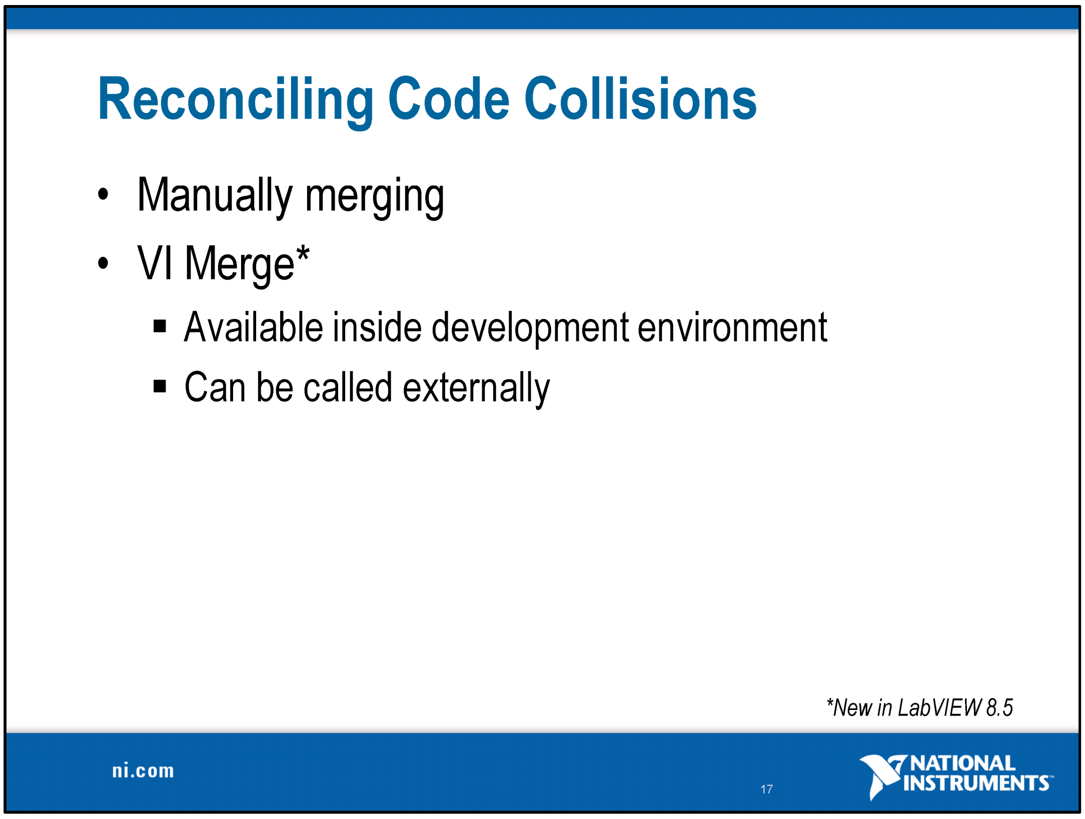


Code Review Benefits

- Bugs identified earlier in the product life-cycle
- Developers improve their programming abilities through feedback
- Additional person is familiar with the program

Recommendations

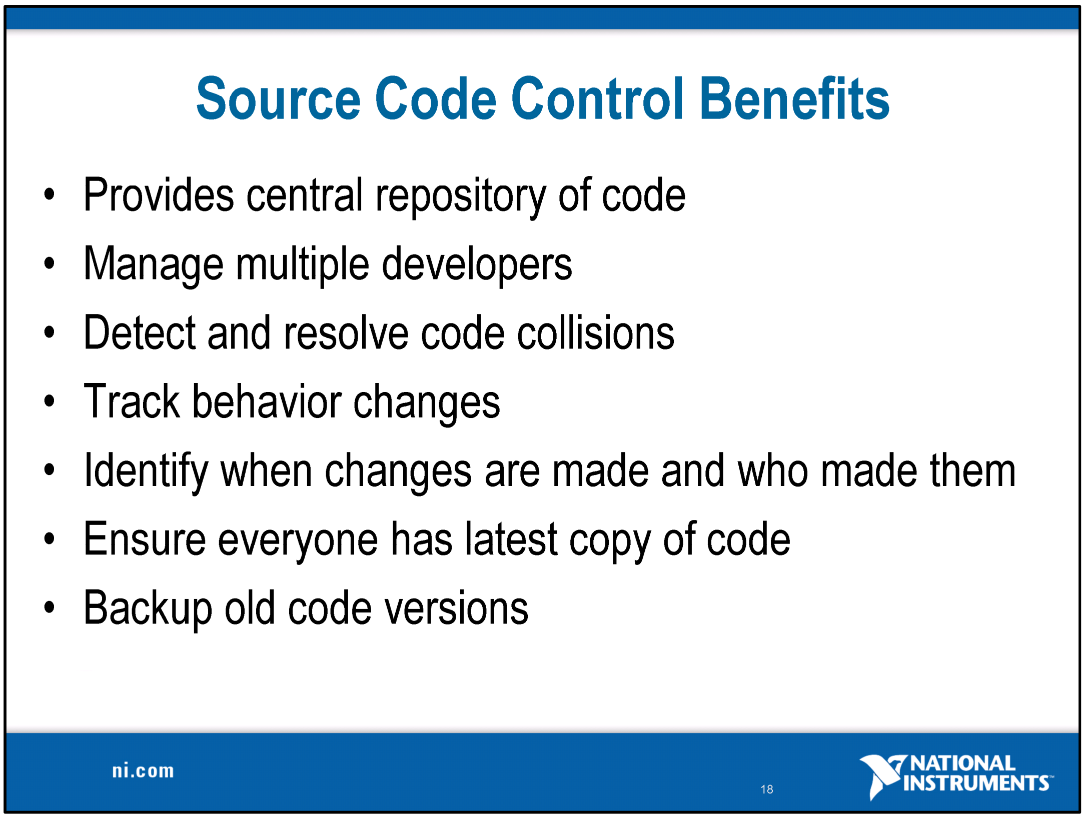
- Use LabVIEW Style Guide as a reference
- Perform regularly during development
- Use VI Analyzer as an aide



Reconciling Code Collisions

- Manually merging
- VI Merge*
  - Available inside development environment
  - Can be called externally

*New in LabVIEW 8.5



Source Code Control Benefits

- Provides central repository of code
- Manage multiple developers
- Detect and resolve code collisions
- Track behavior changes
- Identify when changes are made and who made them
- Ensure everyone has latest copy of code
- Backup old code versions

ni.com 18 NATIONAL INSTRUMENTS™

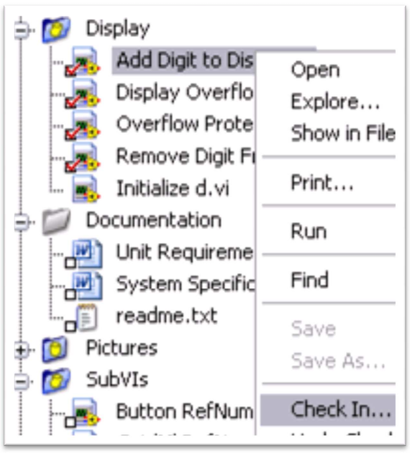
Large applications typically entail the use of more than one developer. This poses a challenge in any programming language.

Setup concrete example. Developer A does ___, Developer B does _____

Consider what happens if two developers modify the same file at the same time or if they make a change to a piece of code that affects code another developer is working on.

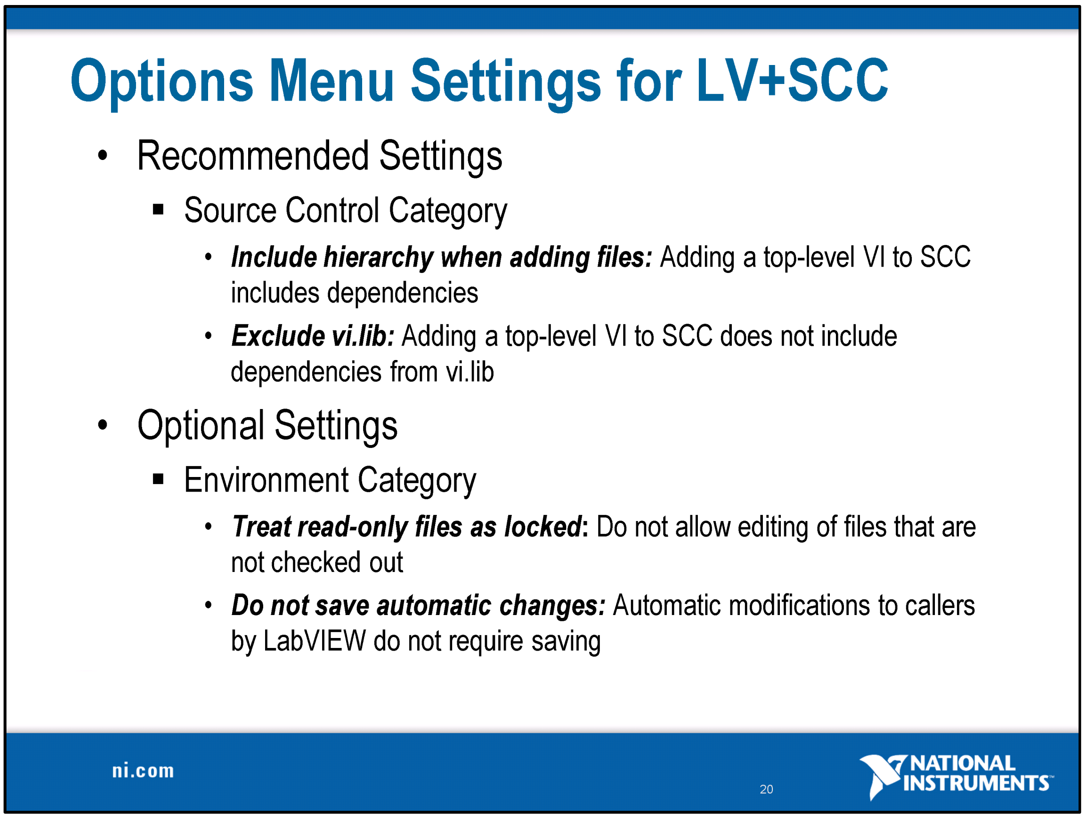
SCC Integration with LabVIEW

- Third-party source control integration with:
  - Microsoft Visual SourceSafe
  - Microsoft Team System
  - Perforce
  - Rational ClearCase
  - PCVS (Serena) Version Manager
  - MKS Source Integrity
  - Seapine Surround SCM
  - Borland StarTeam
  - Telelogic Synergy
  - ionForge Evolution
  - subVersion**
- Access SCC tools via LabVIEW Project
- Project specific settings*



*New in LabVIEW 8.5

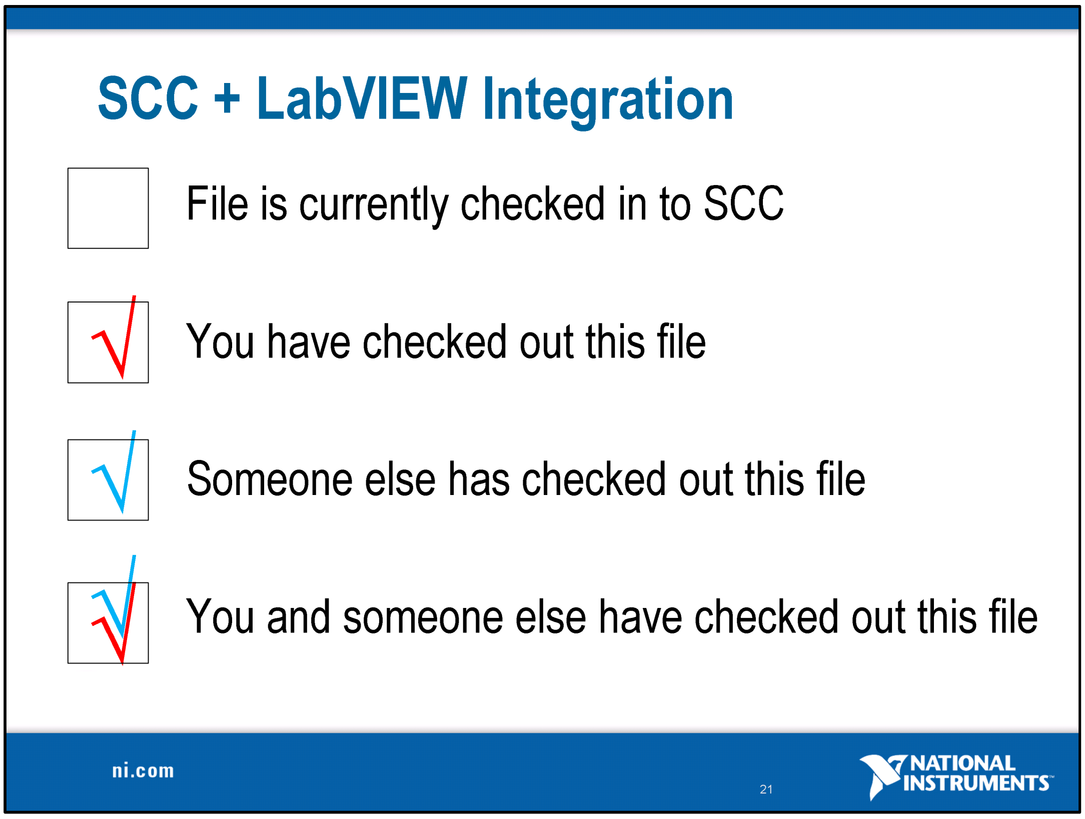
**subVersion is open source and requires plug-in



Options Menu Settings for LV+SCC

- Recommended Settings
  - Source Control Category
    - ***Include hierarchy when adding files:*** Adding a top-level VI to SCC includes dependencies
    - ***Exclude vi.lib:*** Adding a top-level VI to SCC does not include dependencies from vi.lib
- Optional Settings
  - Environment Category
    - ***Treat read-only files as locked***: Do not allow editing of files that are not checked out
    - ***Do not save automatic changes:*** Automatic modifications to callers by LabVIEW do not require saving

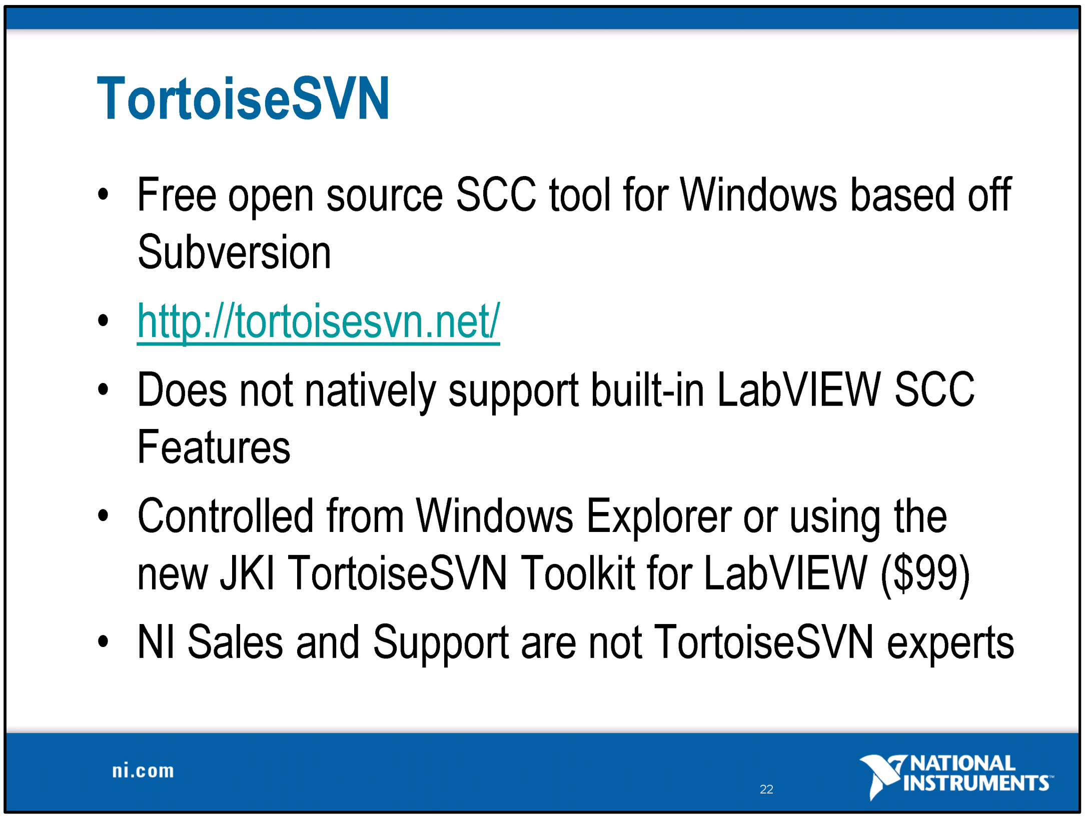
ni.com 20 NATIONAL INSTRUMENTS



SCC + LabVIEW Integration

- ☐ File is currently checked in to SCC
- ☑ You have checked out this file
- ☑ Someone else has checked out this file
- ☑☑ You and someone else have checked out this file

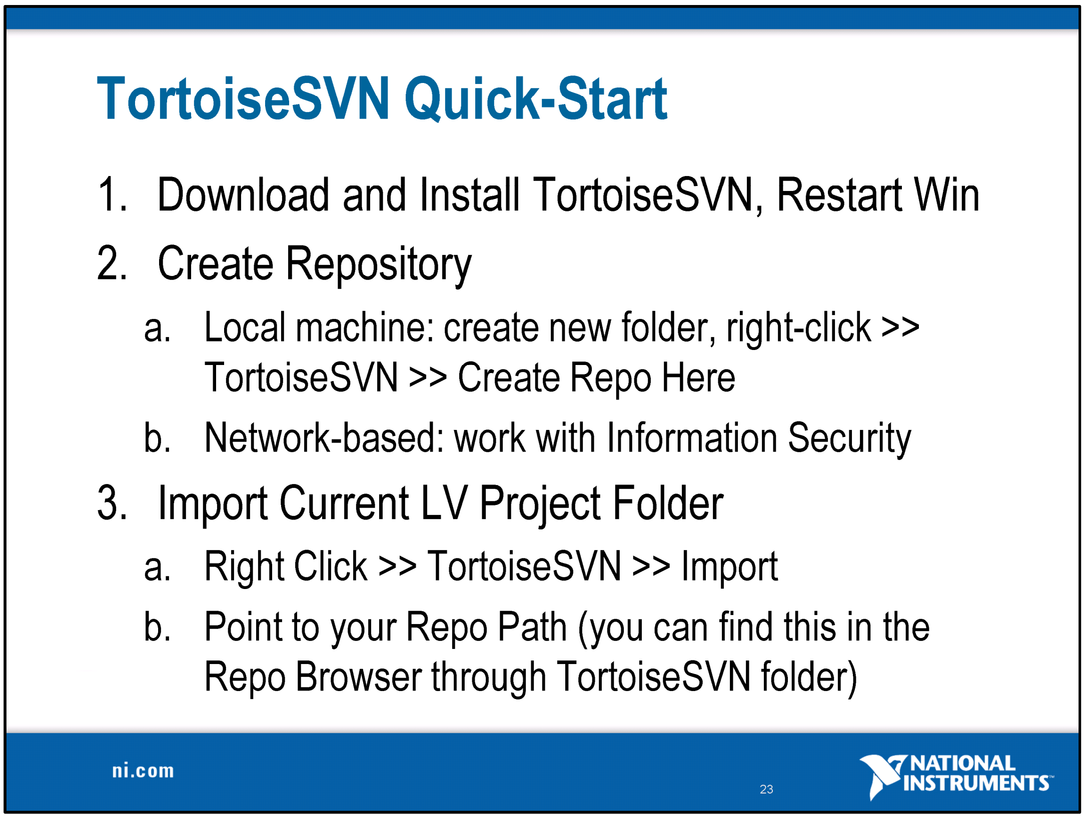
ni.com 21 NATIONAL INSTRUMENTS



TortoiseSVN

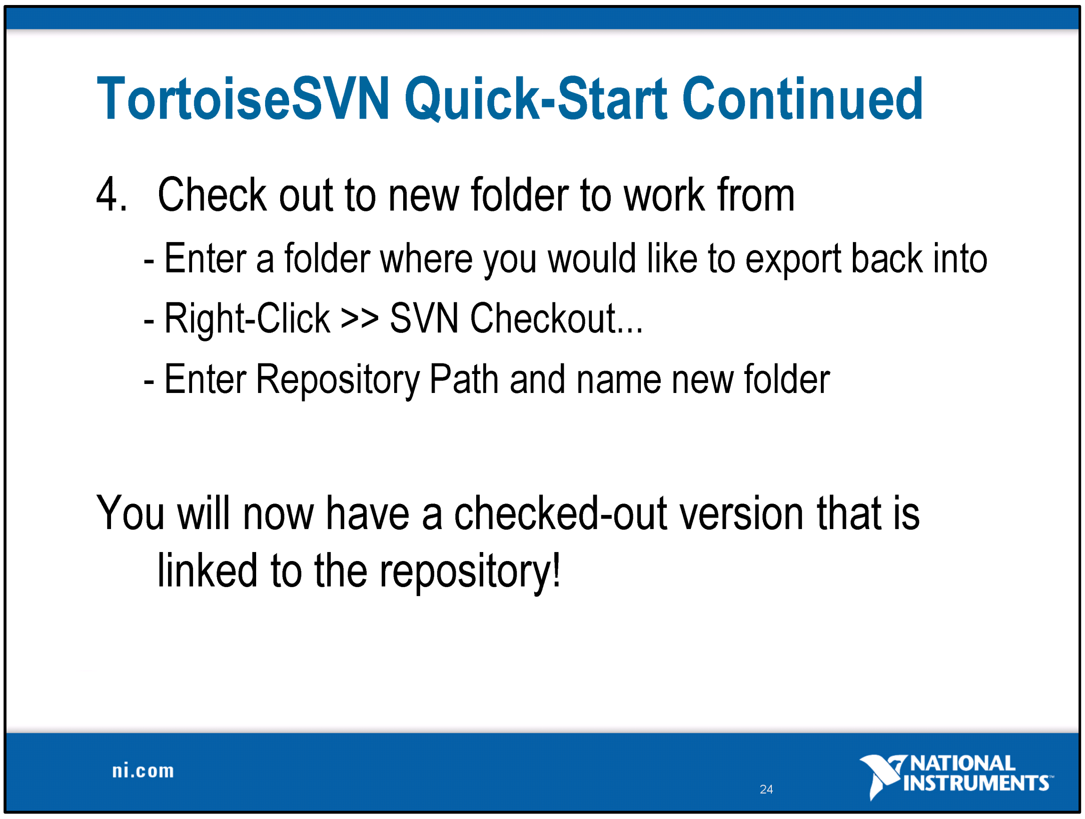
- Free open source SCC tool for Windows based off Subversion
- http://tortoisesvn.net/
- Does not natively support built-in LabVIEW SCC Features
- Controlled from Windows Explorer or using the new JKI TortoiseSVN Toolkit for LabVIEW ($99)
- NI Sales and Support are not TortoiseSVN experts

ni.com 22 NATIONAL INSTRUMENTS



TortoiseSVN Quick-Start

1. Download and Install TortoiseSVN, Restart Win
2. Create Repository
   a. Local machine: create new folder, right-click >> TortoiseSVN >> Create Repo Here
   b. Network-based: work with Information Security
3. Import Current LV Project Folder
   a. Right Click >> TortoiseSVN >> Import
   b. Point to your Repo Path (you can find this in the Repo Browser through TortoiseSVN folder)



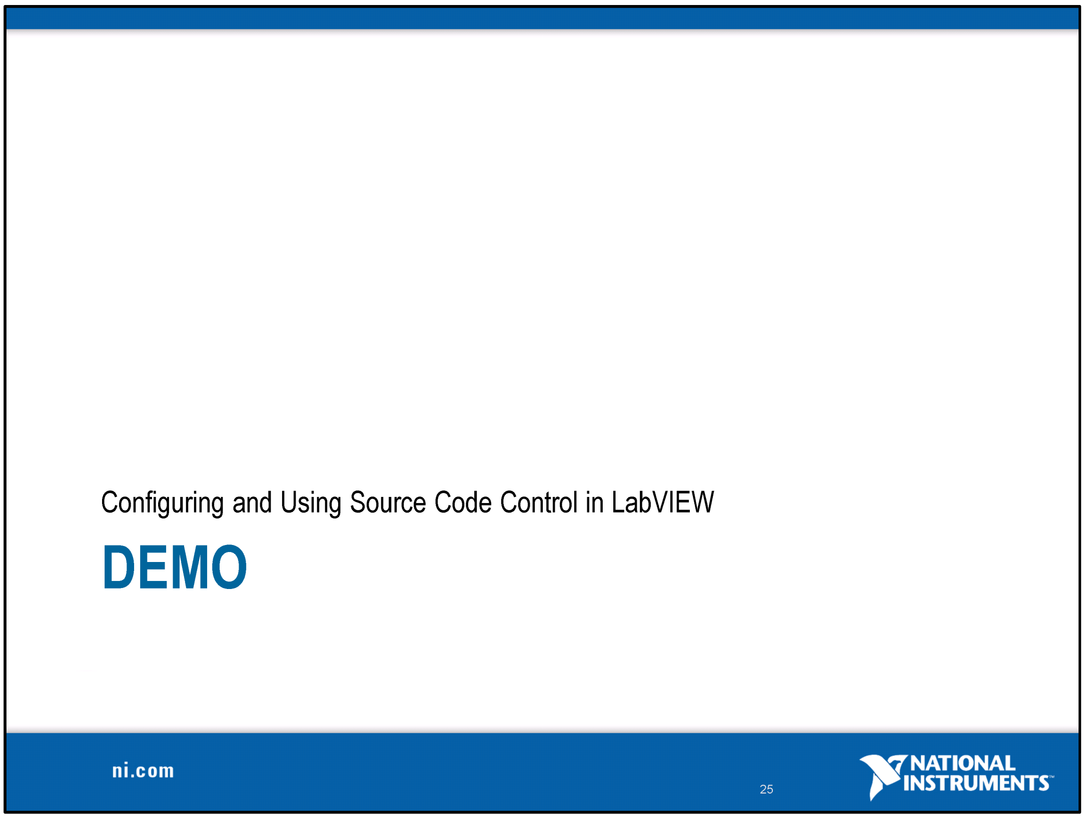
TortoiseSVN Quick-Start Continued

4. Check out to new folder to work from
   - Enter a folder where you would like to export back into
   - Right-Click >> SVN Checkout...
   - Enter Repository Path and name new folder

You will now have a checked-out version that is linked to the repository!

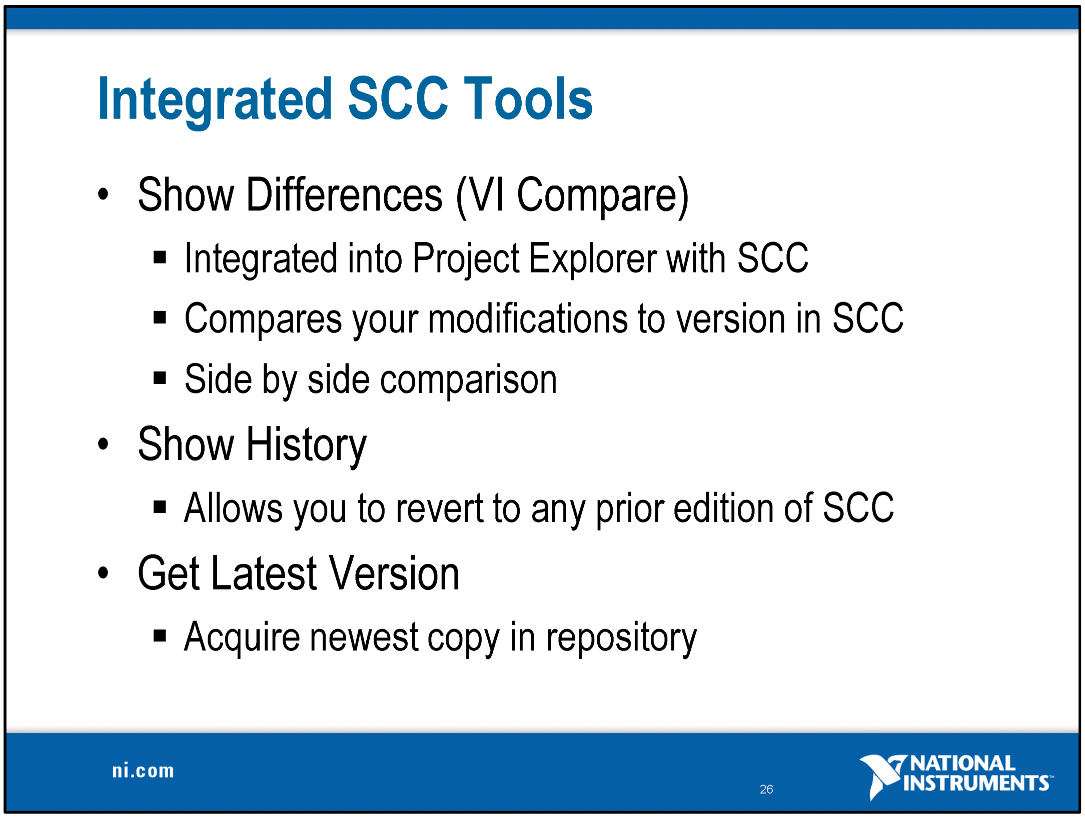
ni.com

24



Configuring and Using Source Code Control in LabVIEW

DEMO



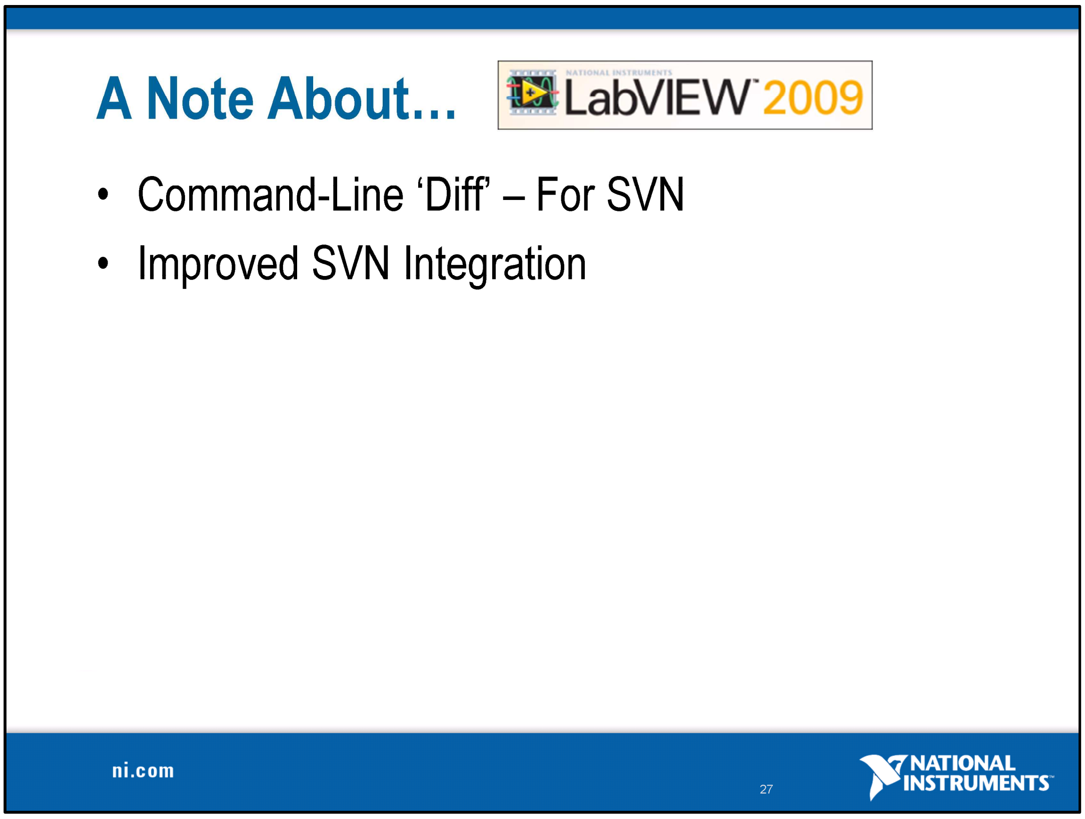
Integrated SCC Tools

- Show Differences (VI Compare)
  - Integrated into Project Explorer with SCC
  - Compares your modifications to version in SCC
  - Side by side comparison
- Show History
  - Allows you to revert to any prior edition of SCC
- Get Latest Version
  - Acquire newest copy in repository

ni.com

26

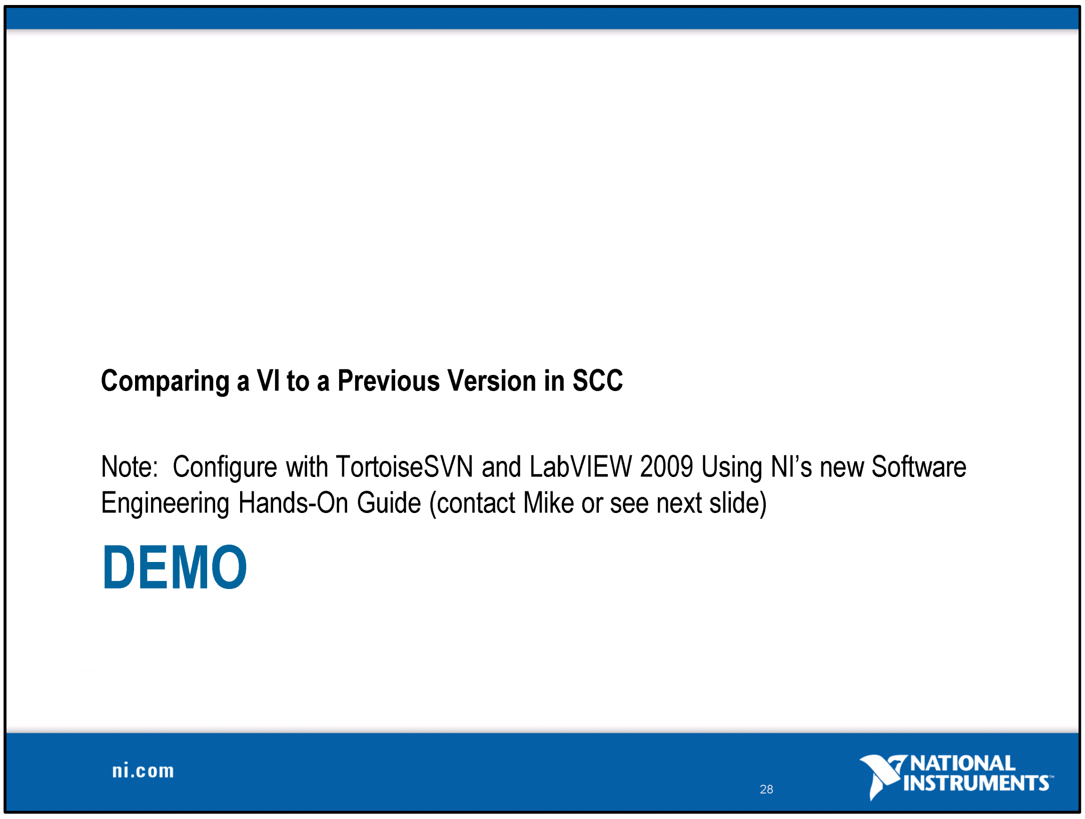
NATIONAL INSTRUMENTS



A Note About… LabVIEW 2009

- Command-Line 'Diff' – For SVN
- Improved SVN Integration

ni.com 27 NATIONAL INSTRUMENTS



Comparing a VI to a Previous Version in SCC

Note: Configure with TortoiseSVN and LabVIEW 2009 Using NI's new Software Engineering Hands-On Guide (contact Mike or see next slide)

DEMO

ni.com

28

NATIONAL INSTRUMENTS™

TortoiseSVN Diff Setup

Ensure that LabVIEW and TortoiseSVN are installed

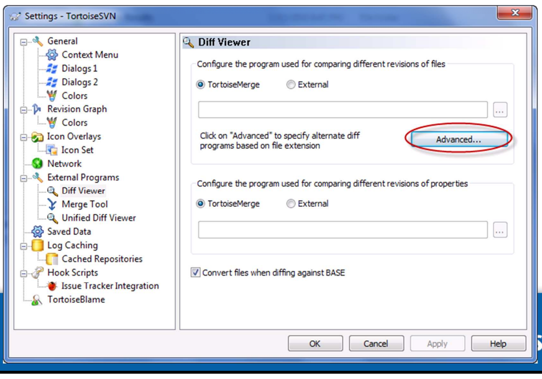
If using LabVIEW 2009, enter ‘skipSVNFolders=true’ into LabVIEW.ini token

Make sure TortoiseSVN is calling LVCompare.exe for graphical differencing

- Right click in Windows Explorer
- Select **TortoiseSVN > Settings**
- Select **Advanced** and enter the following for a .vi file type (this can also be used for a .ctl)

"C:\Program Files\National Instruments\Shared\LabVIEW Compare\LVCompare.exe" %mine %base

See flags on next page.
You can add these to your LVCompare call



Settings - TortoiseSVN

Diff Viewer

Configure the program used for comparing different revisions of files

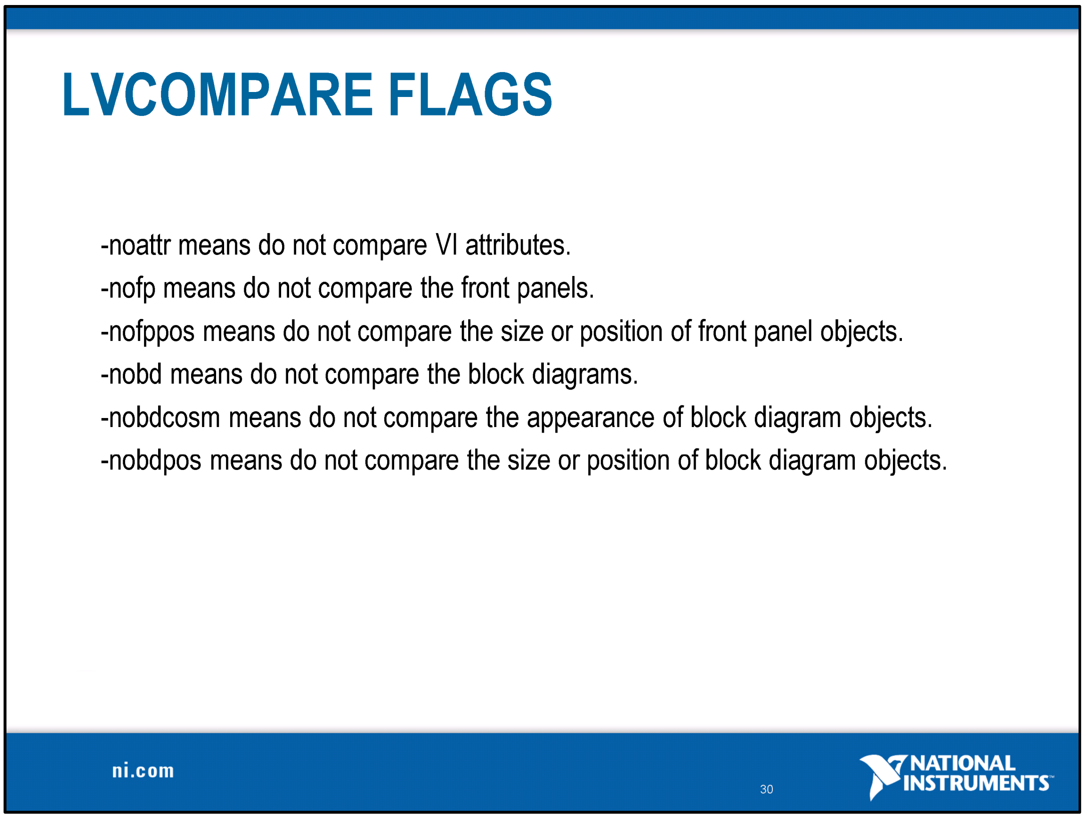
[✓ TortoiseMerge] External

Click on "Advanced" to specify alternate diff programs based on file extension

Configure the program used for comparing different revisions of properties

[✓ TortoiseMerge] External

[✓] Convert files when diffing against BASE



LVCOMPARE FLAGS

-noattr means do not compare VI attributes.

-nofp means do not compare the front panels.

-nofppos means do not compare the size or position of front panel objects.

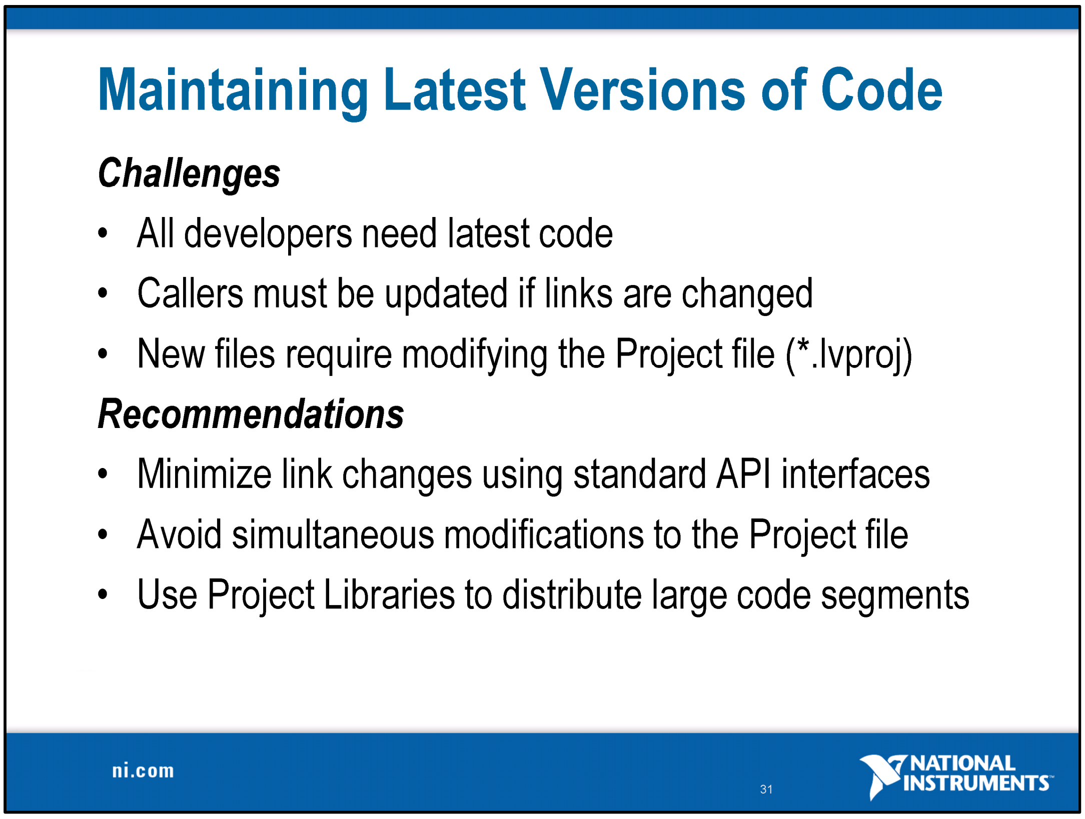
-nobd means do not compare the block diagrams.

-nobdcosm means do not compare the appearance of block diagram objects.

-nobdpos means do not compare the size or position of block diagram objects.

ni.com

30



Maintaining Latest Versions of Code

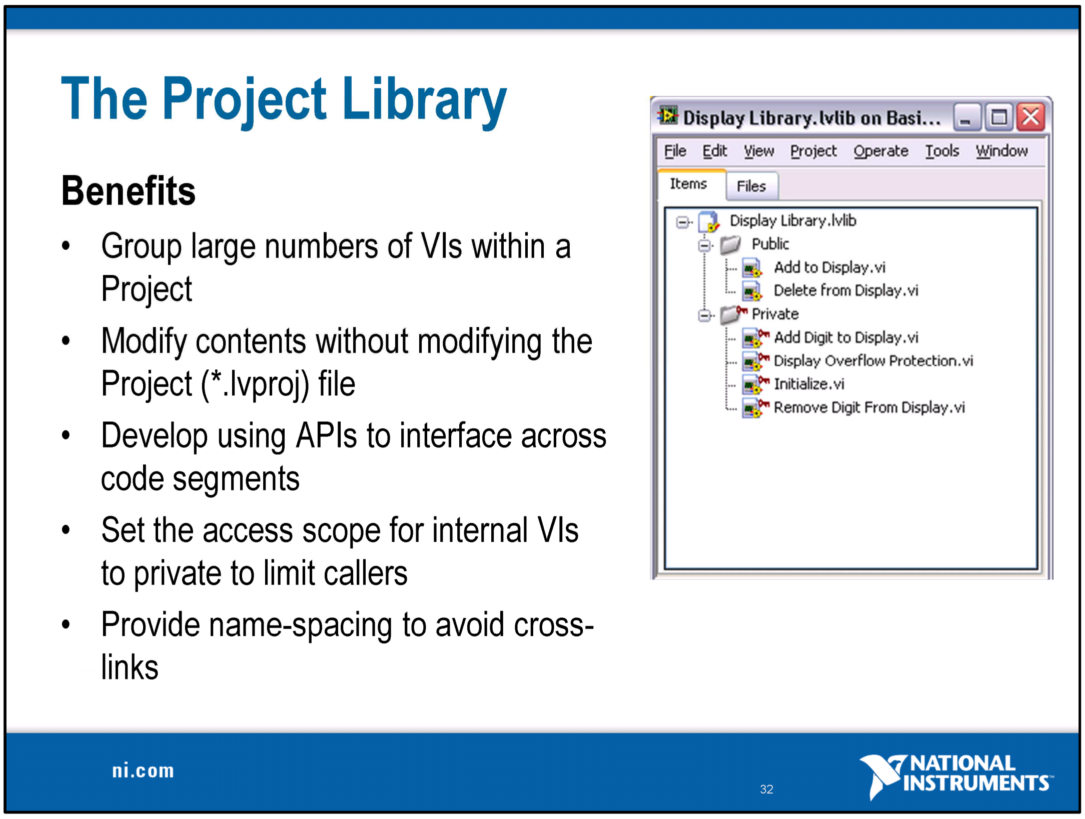
Challenges

- All developers need latest code
- Callers must be updated if links are changed
- New files require modifying the Project file (*.lvproj)

Recommendations

- Minimize link changes using standard API interfaces
- Avoid simultaneous modifications to the Project file
- Use Project Libraries to distribute large code segments

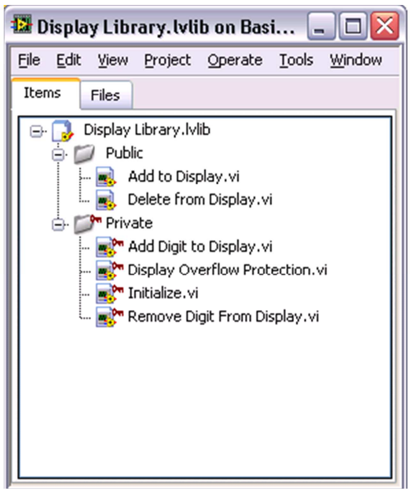
ni.com 31 NATIONAL INSTRUMENTS



The Project Library

Benefits

- Group large numbers of VIs within a Project
- Modify contents without modifying the Project (*.lvproj) file
- Develop using APIs to interface across code segments
- Set the access scope for internal VIs to private to limit callers
- Provide name-spacing to avoid cross-links


Display Library.lvlib on Basi...
File Edit View Project Operate Tools Window
Items
Files
Display Library.lvlib
Public
Add to Display.vi
Delete from Display.vi
Private
Add Digit to Display.vi
Display Overflow Protection.vi
Initialize.vi
Remove Digit From Display.vi

ni.com

32

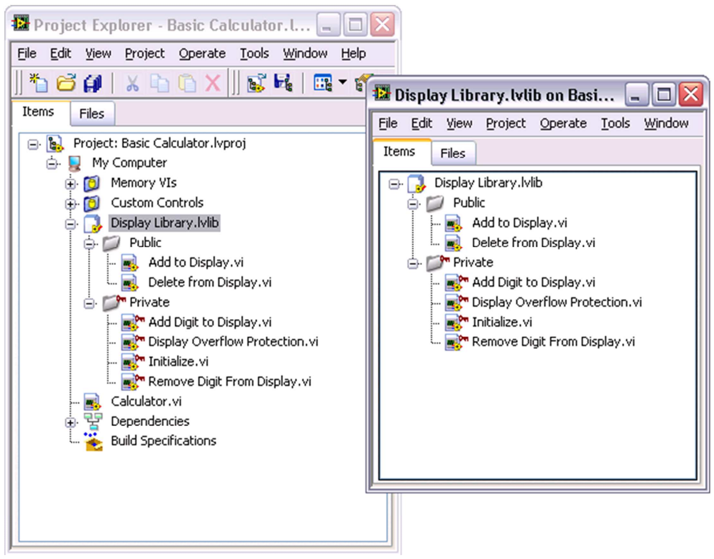
The project originally came out in LabVIEW 8, but we've made a lot of changes as a result of customer feedback in LabVIEW 8.5

"So if you haven't looked lately, look again"

> The Project Explorer was introduced in LabVIEW 8.x to address many of the challenges mentioned on the previous slide.
> Review benefits on slide

Project Libraries

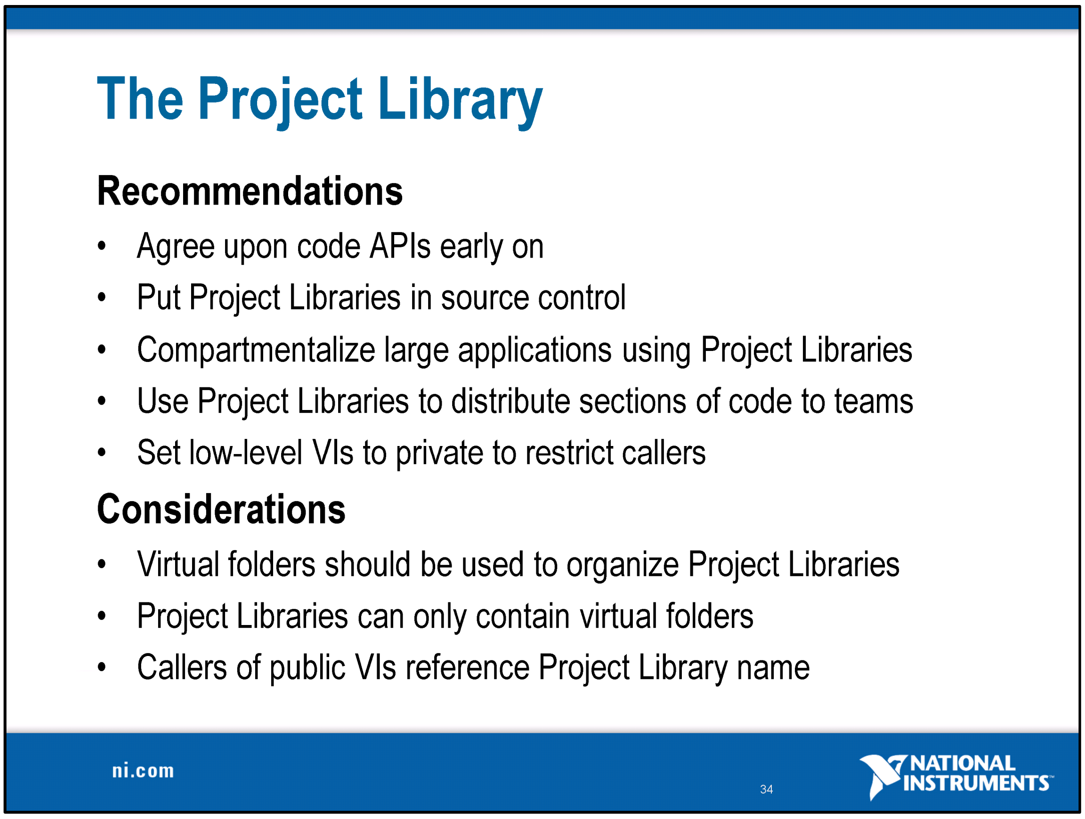
DEMO


Project Explorer - Basic Calculator.l...
File Edit View Project Operate Tools Window Help
Items Files

Project: Basic Calculator.lvproj
- My Computer
  - Memory VIs
  - Custom Controls
  - Display Library.lvlib
    - Public
      - Add to Display.vi
      - Delete from Display.vi
    - Private
      - Add Digit to Display.vi
      - Display Overflow Protection.vi
      - Initialize.vi
      - Remove Digit From Display.vi
  - Calculator.vi
  - Dependencies
  - Build Specifications

Display Library.lvlib on Basi...
File Edit View Project Operate Tools Window
Items Files

Display Library.lvlib
- Public
  - Add to Display.vi
  - Delete from Display.vi
- Private
  - Add Digit to Display.vi
  - Display Overflow Protection.vi
  - Initialize.vi
  - Remove Digit From Display.vi



The Project Library

Recommendations

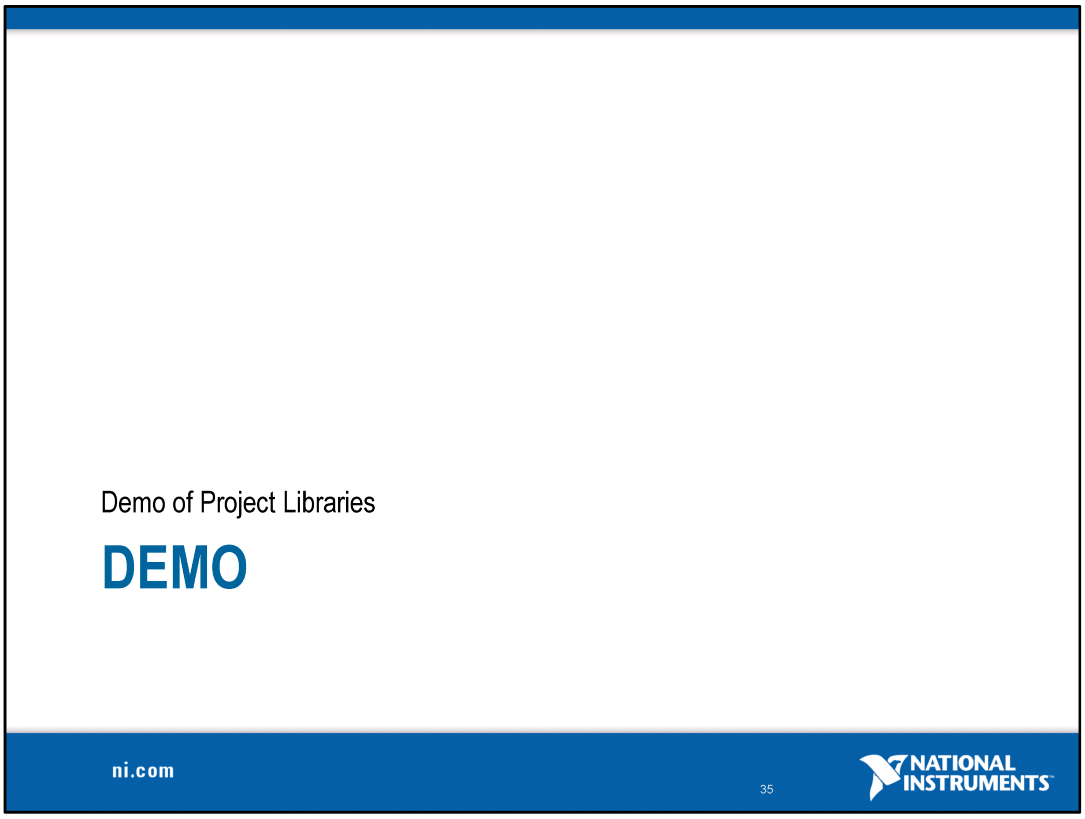
- Agree upon code APIs early on
- Put Project Libraries in source control
- Compartmentalize large applications using Project Libraries
- Use Project Libraries to distribute sections of code to teams
- Set low-level VIs to private to restrict callers

Considerations

- Virtual folders should be used to organize Project Libraries
- Project Libraries can only contain virtual folders
- Callers of public VIs reference Project Library name

ni.com

34

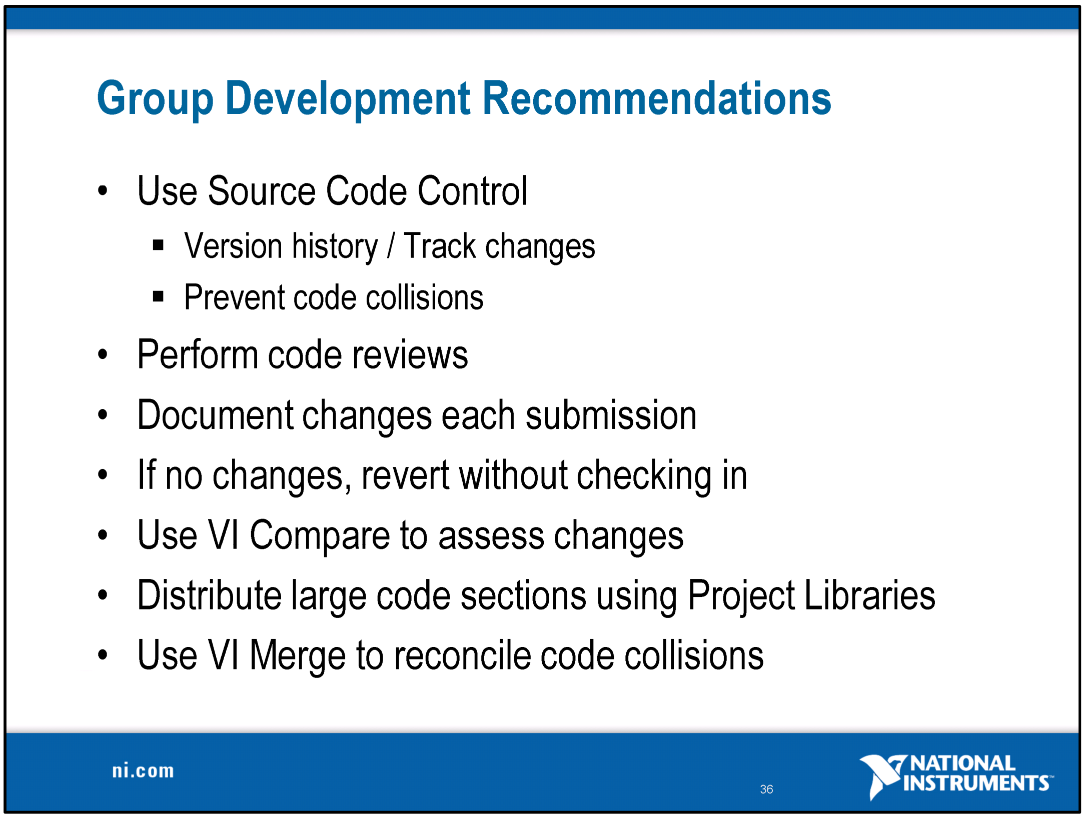


Demo of Project Libraries

DEMO

ni.com

35

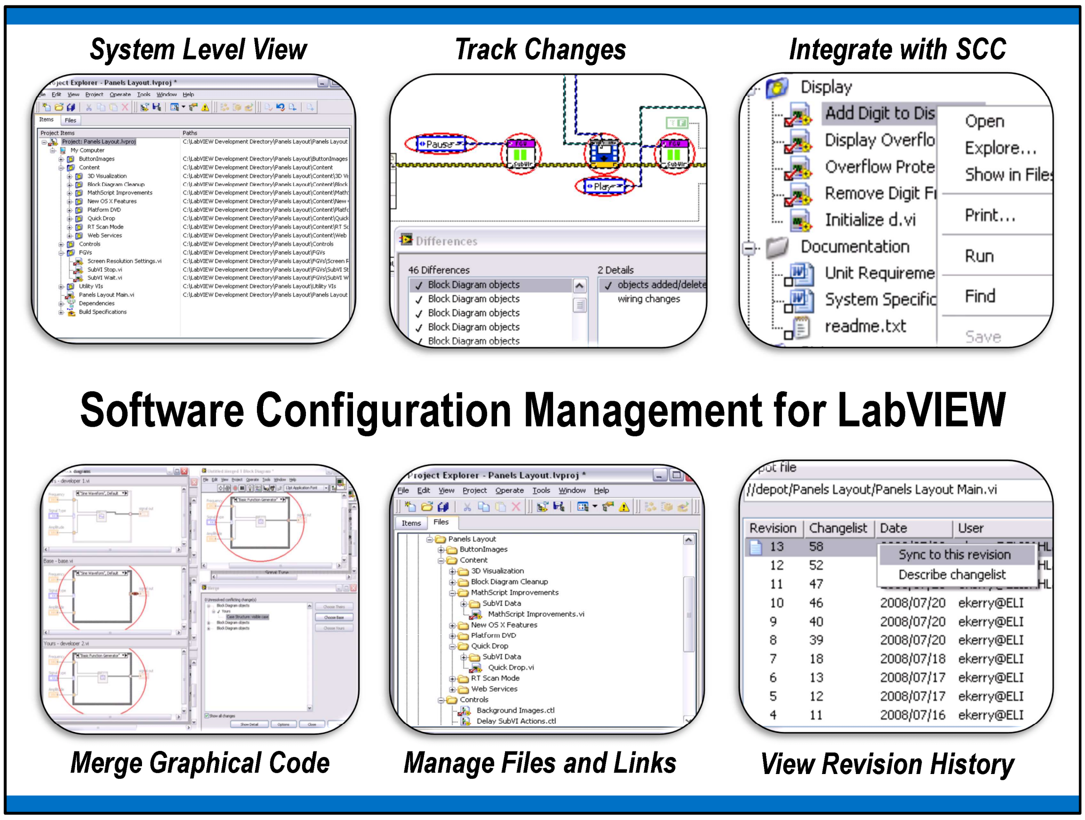


Group Development Recommendations

- Use Source Code Control
  - Version history / Track changes
  - Prevent code collisions
- Perform code reviews
- Document changes each submission
- If no changes, revert without checking in
- Use VI Compare to assess changes
- Distribute large code sections using Project Libraries
- Use VI Merge to reconcile code collisions

ni.com

36

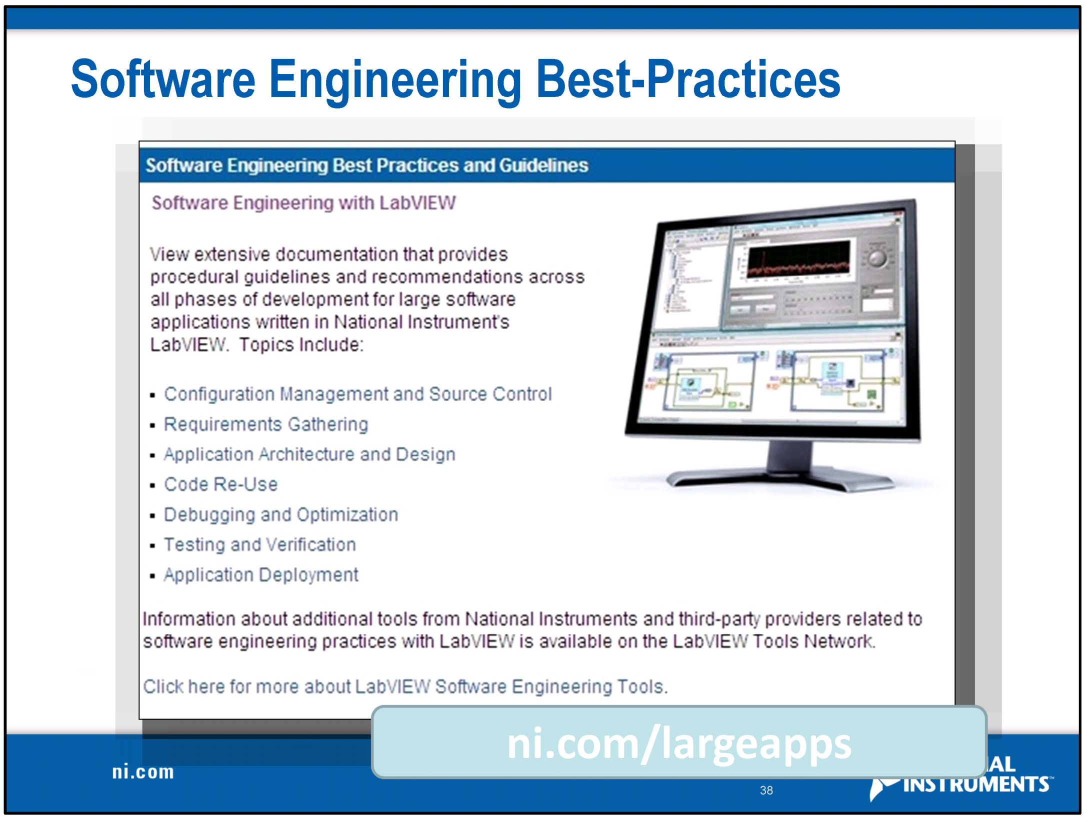


//depot/Panels Layout/Panels Layout Main.vi

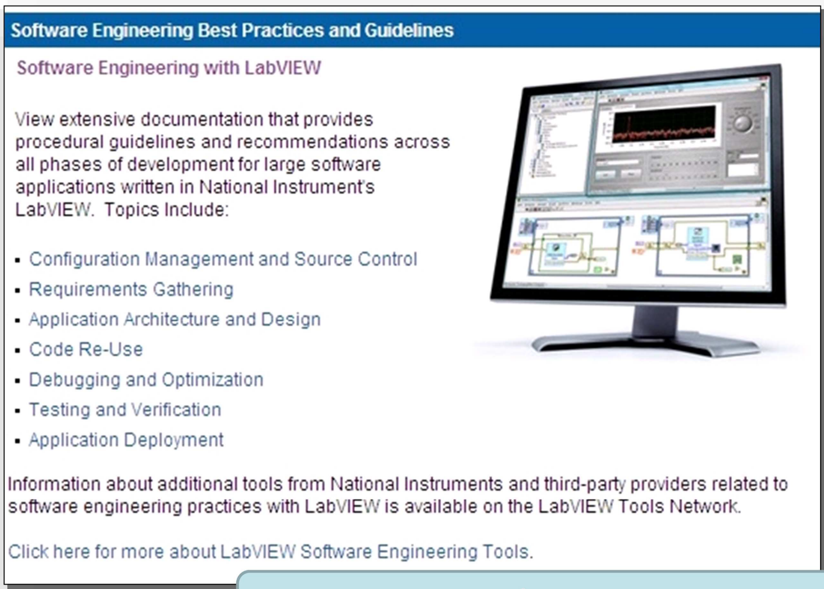
Revision	Changelist	Date	User
13	58	[illegible]	[illegible]
12	52	[illegible]	[illegible]
11	47	[illegible]	[illegible]
10	46	2008/07/20	ekerry@ELI
9	40	2008/07/20	ekerry@ELI
8	39	2008/07/20	ekerry@ELI
7	18	2008/07/18	ekerry@ELI
6	13	2008/07/17	ekerry@ELI
5	12	2008/07/17	ekerry@ELI
4	11	2008/07/16	ekerry@ELI

Note: this is not an exhaustive list, rather it is a way to demonstrate how some common configuration management tasks are performed in LabVIEW.

1. **System Level View –** The project explorer in LabVIEW provides developers a system level view of all their resources, including source code, any affiliated files or documentation and even hardware resources, including different platform targets such as RT or FPGA
2. **Track Changes** – You can see in this screen shot how LabVIEW can actually indicate exactly what aspects of a block diagram or front panel have been changed or modified. This is made possible thanks to graphical differencing
3. **Integration with SCC –** The ability to integrate with SCC from within LabVIEW is advantageous for developers because they can see the current status of files and access generic SCC functionality without leaving LabVIEW
4. **Merge Graphical Code –** an important tool for group development requires the ability to combine changes made by multiple developers. This is now possible in LabVIEW 8.5
5. **Manage Files and Links –** there are many tools in LabVIEW that have been introduced to address common challenges regarding file management. Moving or renaming files can break links or even cause incorrect links to dependencies on disk. LabVIEW actually allows you to perform these file operations from within the Project in order to preserve correct links and avoid potential cross-link scenarios.
6. **View Revision History** – when integration with SCC is turned on you can view all the revisions that have been submitted and stored in the SCC repository. You can even right click on the particular revision and select ‘sync to this revision,’ which pulls down that version of the file. This view also shows and commonts or information about the file from the time when it was checked in.



Software Engineering Best-Practices



Software Engineering Best Practices and Guidelines

Software Engineering with LabVIEW

View extensive documentation that provides procedural guidelines and recommendations across all phases of development for large software applications written in National Instrument's LabVIEW. Topics Include:

- Configuration Management and Source Control
- Requirements Gathering
- Application Architecture and Design
- Code Re-Use
- Debugging and Optimization
- Testing and Verification
- Application Deployment

Information about additional tools from National Instruments and third-party providers related to software engineering practices with LabVIEW is available on the LabVIEW Tools Network.

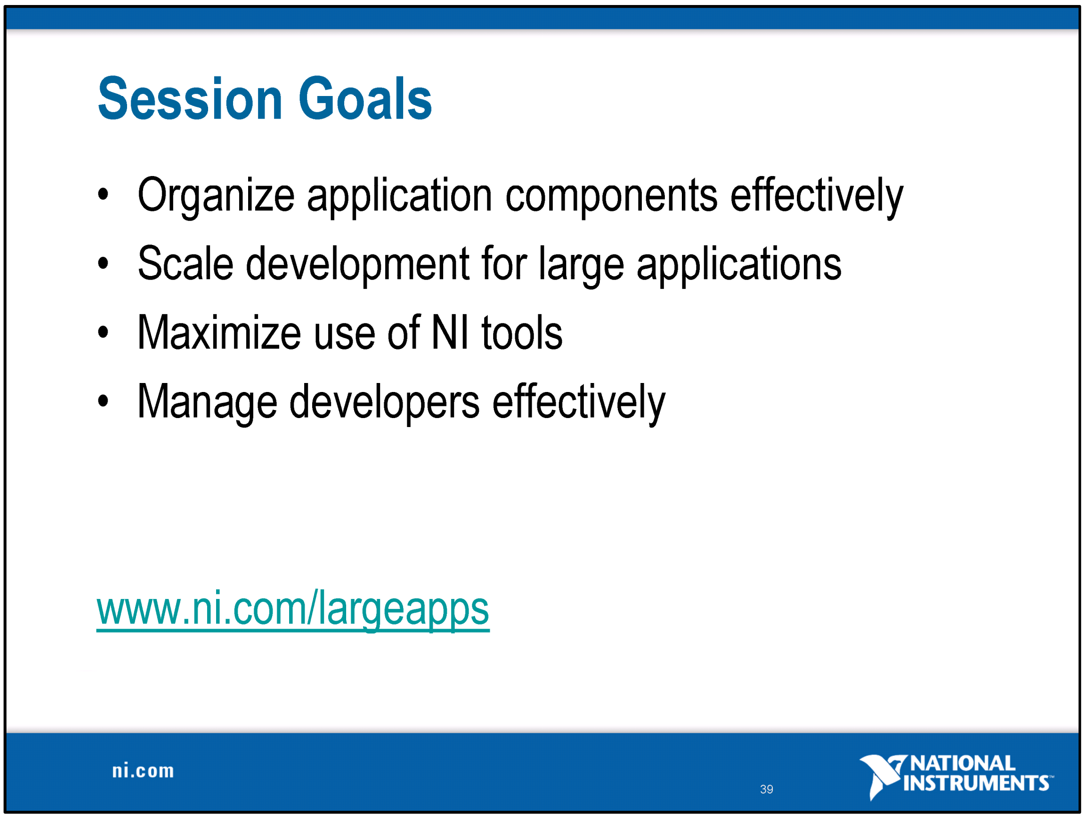
Click here for more about LabVIEW Software Engineering Tools.

ni.com/largeapps

ni.com

38

This is the questions slide – keep this up to encourage them all to write down the URL at the top of the screen



Session Goals

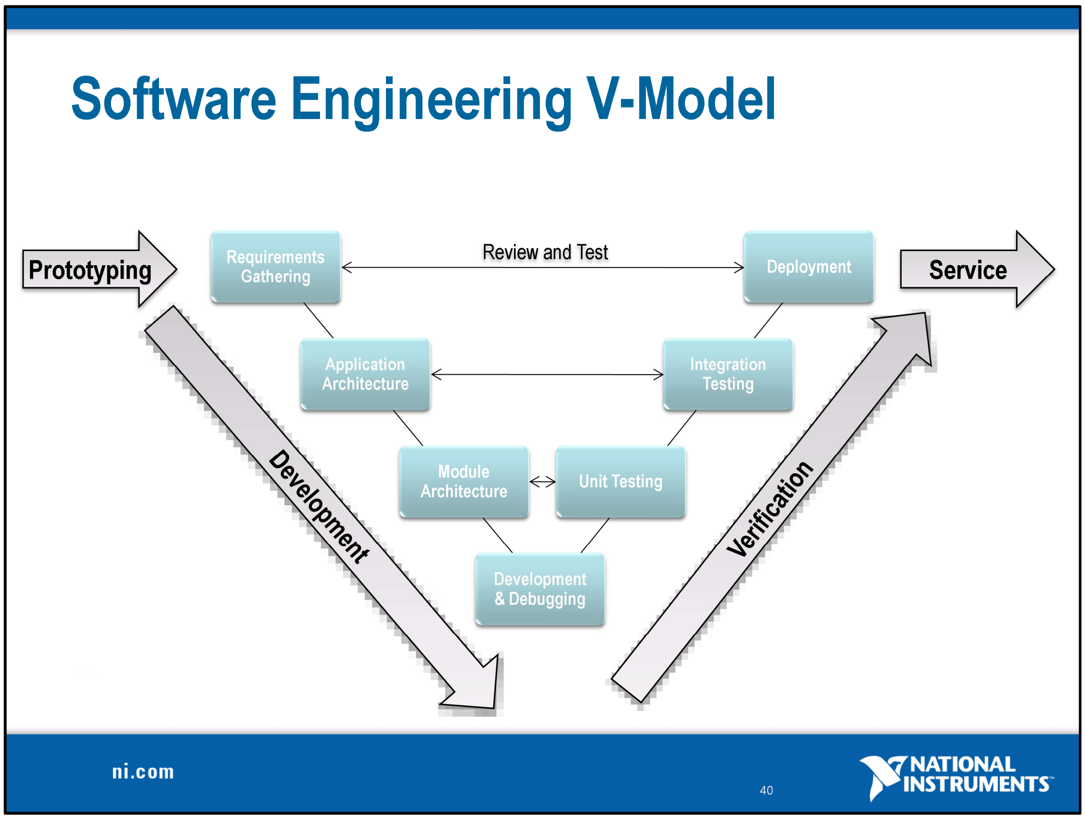
- Organize application components effectively
- Scale development for large applications
- Maximize use of NI tools
- Manage developers effectively

www.ni.com/largeapps

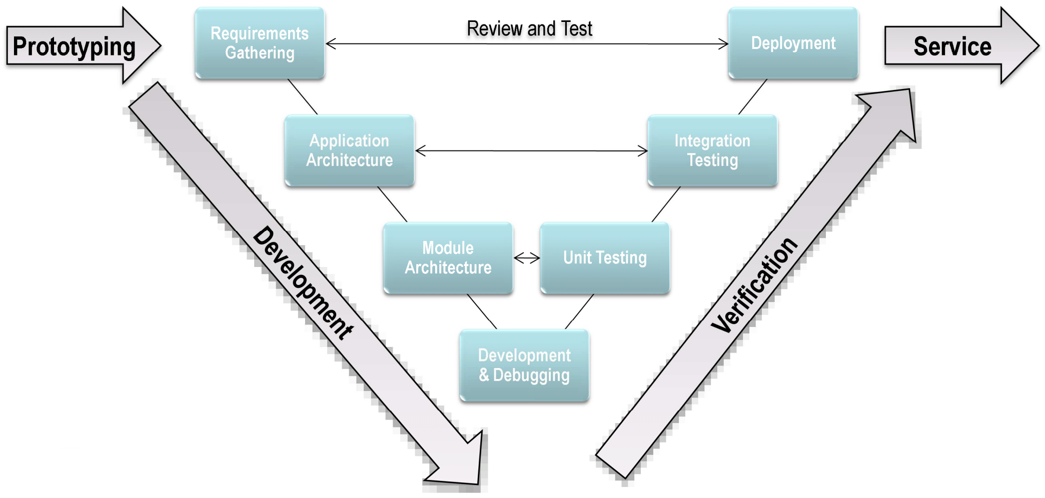
ni.com 39 NATIONAL INSTRUMENTS

The content provided in this session is intended to help attendees understand how to best organize the various components of their application such that development practices will scale for large numbers of VIs and numerous developers. In the process, we will discuss the proper techniques for using NI tools and in particular, the Project Explorer

The attendee should walk away from this session with a clear understanding of how they will achieve these goals using the recommendations you provide



Software Engineering V-Model



ni.com

40

NATIONAL INSTRUMENTS

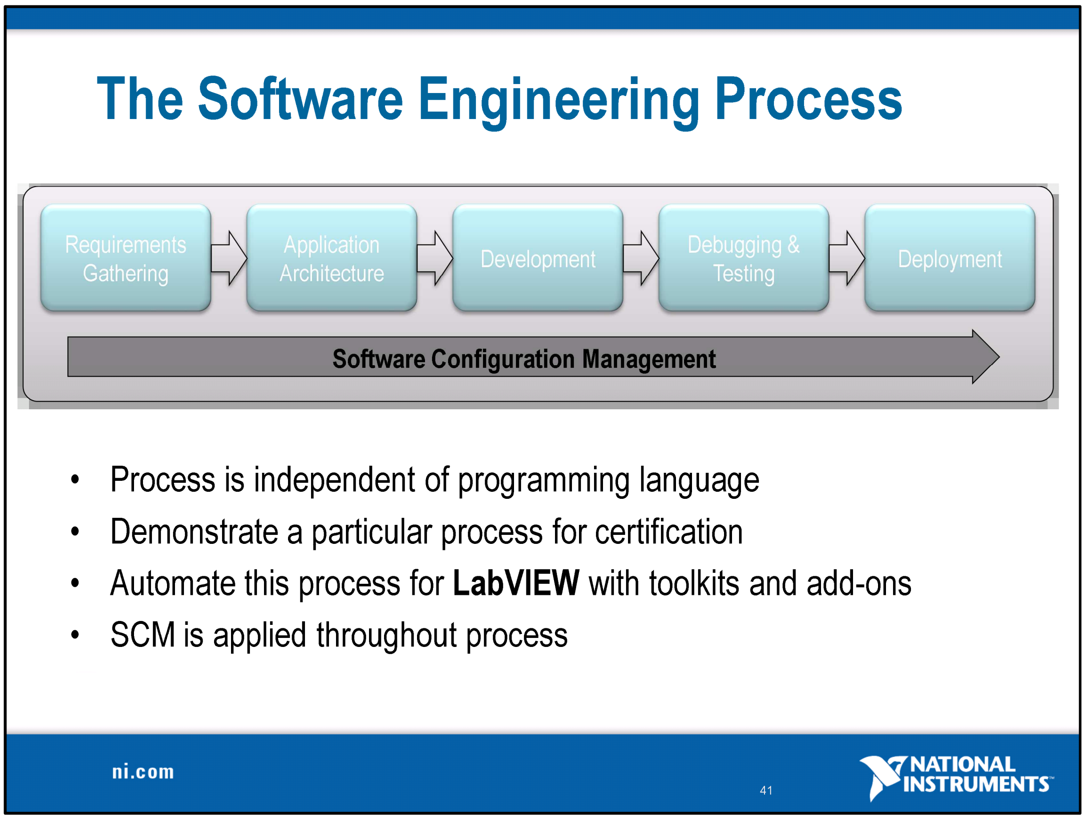
Software engineering typically refers to a regimented and procedural methodology for developing software. As software has gotten more complex and team sizes have grown, the software engineering process model has evolved to encourage efficient development that ensures quality and meets the expectations of the end-user.

A process model prescribes specific phases and tasks to accomplish during the development process. While there are many representations of the software engineering ‘process,’ including the waterfall model, the spiral process model, or more contemporary models such as agile or XP (extreme programming) models, perhaps one of the most common and rigorous models is shown in this diagram: the ‘v-model.’

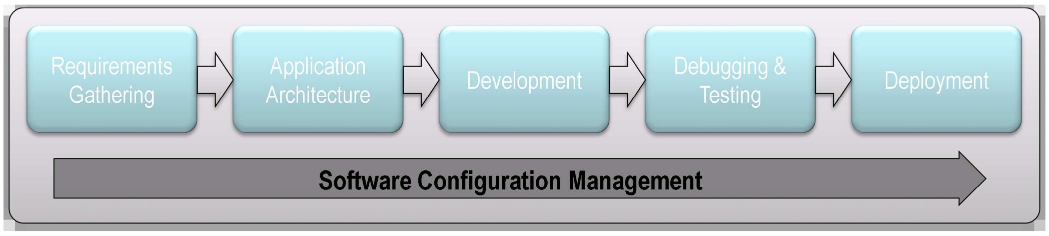
If you look closely at this diagram you’ll notice that it begins with the defining of requirements for the product. These are often high level and are often developed through discussion with the customer or end-user.

One thing to realize is that the level of criticality directly impacts how granular you have to be in this process. You may not need to spend any formal time architecting your application or the specific modules of code, but these are typically very important for large, complex systems.

There are several SEP models, such as Waterfall, Spiral and Agile, but they all share common ground in terms of having these various phases and tasks that have to be completed. Many companies today are leaning more towards increasingly agile methods. Basically, this means that for many it’s unrealistic to abide by a waterfall method, in which you work on a particular phase, then agree that you’re done and move on. In reality, complex applications require a lot of back and forth between these different phases. For example, something in the prototype stage doesn’t work as expected and therefore require a new definition for how a particular part of the application will operate, thereby impacting requirements documentation and other components such as the test framework. We’ll look at how to address some of these common challenges in just a few slides..



The Software Engineering Process



- Process is independent of programming language
- Demonstrate a particular process for certification
- Automate this process for **LabVIEW** with toolkits and add-ons
- SCM is applied throughout process

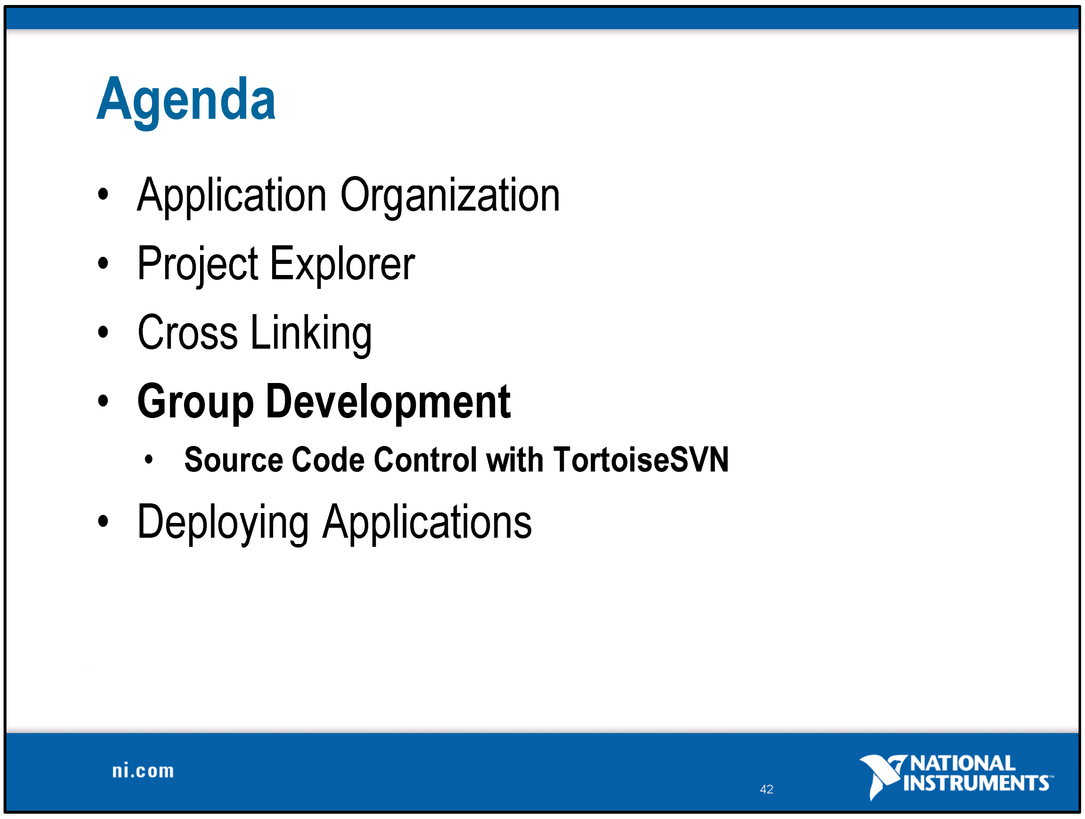
ni.com 41 NATIONAL INSTRUMENTS

This diagram is a slightly different and slightly simpler way of viewing this process. It reflects what is typically referred to as the waterfall method of development. While great in principle, most software engineers accept that the waterfall method is impractical and that reality requires significant overlap between these various phases. In other words, it's almost impossible to avoid changes to requirements later in development. The key is to have tools and practices to mitigate the risks caused by these last minute changes and to understand how these changes will impact other aspects of your application.

For the rest of this presentation we're going to be looking at these different phases of development and examining what some of the more common challenges are and how to address them when developing applications with LabVIEW. For those of you who come from backgrounds where you were using a different language, you probably have been through this process before. It's important to realize that the process and many of the methodologies are the same, but that we have tools available that are specific to graphical development in LabVIEW.

LabVIEW is great for rapid prototyping, and it's easy to get started. Consequently it's easy for people to get off on the wrong foot, jump directly into development and skip some of the initial phases such as requirements gathering and application architecture.

It's important to note that configuration management is shown as a part of the entire process. This refers to tools for managing software and is also often referred to as source code control (SCC). We're going to begin by looking at SCM and examining some tools and how they can best be used with LabVIEW.


Agenda

- Application Organization
- Project Explorer
- Cross Linking
- **Group Development**
  - **Source Code Control with TortoiseSVN**
- Deploying Applications

ni.com

42

Defining and Organizing an Application

Best Practices for Using the Project Explorer

Managing Files

Project Tools

Target Management

Deploying Applications

Configuration Management Guidelines

Source Code Control Integration

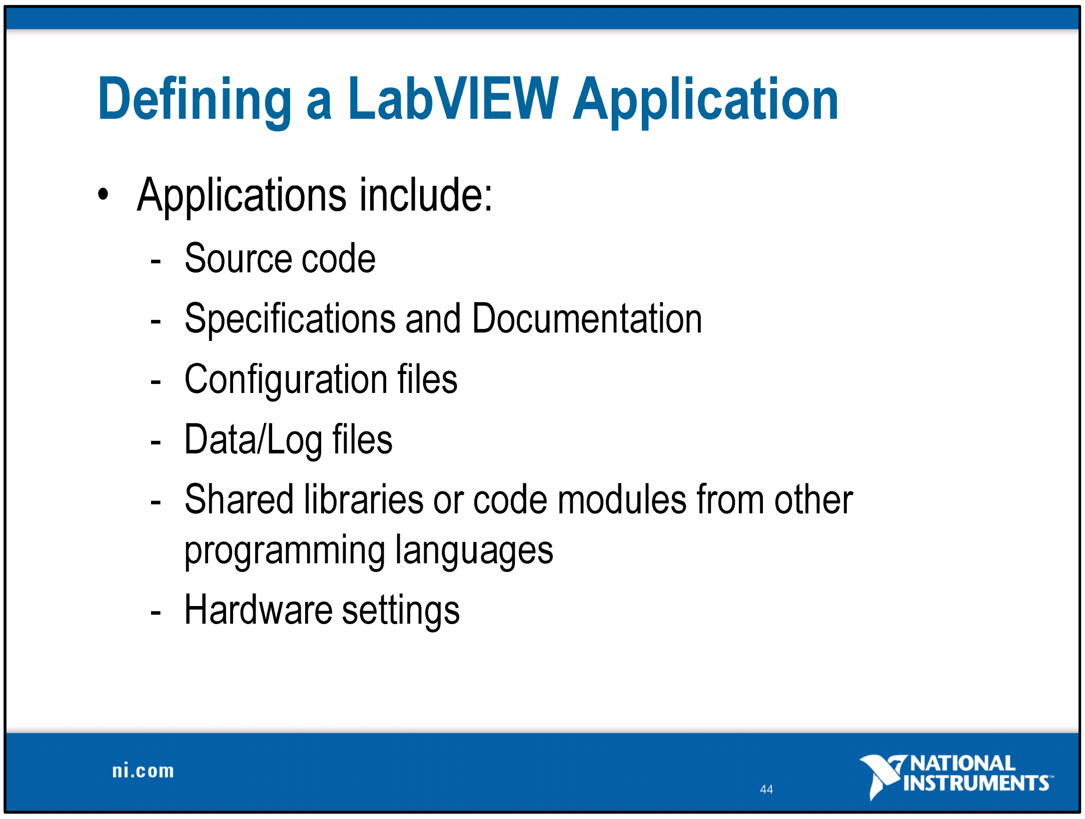
Group Development Practices

Frequently Asked Questions

Organizing and Managing LabVIEW Applications

ni.com

LabVIEW applications are traditionally small and often intended for use as a rapid prototype or a simple program where the developer is the user; however, we're seeing an increasing number of large application being developed in LabVIEW that consist of hundreds if not thousands of VIs and involve multiple developers. In order to ensure success with these applications it is critical to give proper consideration to how files will be organized in your application in order to help you and other developers locate and access application components.



Defining a LabVIEW Application

- Applications include:
  - Source code
  - Specifications and Documentation
  - Configuration files
  - Data/Log files
  - Shared libraries or code modules from other programming languages
  - Hardware settings

ni.com

44

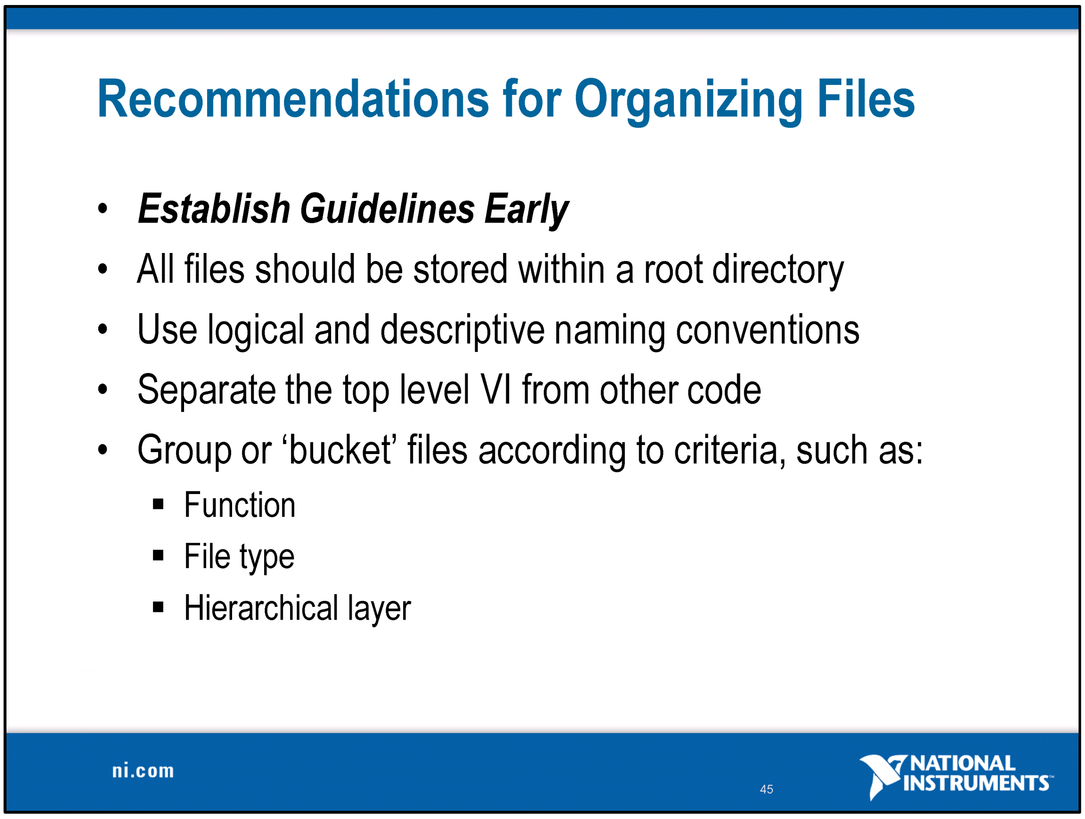
Defining a LabVIEW Application
LabVIEW is often regarded as a tool for rapidly prototyping small applications or automated test applications, many of which are fairly simple applications that rely upon a small number of VIs. However, the complexity and scope of applications built in LabVIEW is constantly increasing, which requires more code, more complex code, and various other resources aside from VIs. For those of you familiar with LabVIEW development prior to LabVIEW 8, you're likely to agree that it becomes difficult to manage your VIs if your application takes on a larger size. Large applications may even frequently consist of smaller components which are in and of themselves 'applications.' We'll discuss this more in a moment. The fact is that as the size and complexity of applications increases, developers need more sophisticated methods and practices for managing application development.
Come to a consensus with the audience on the definition of an 'application.'

Though the term may seem obvious, point out that it encompasses much more than just the code. Most commonly, applications refer to:

Source code
Specifications and Documentation
Configuration files
Data/Log files
Shared libraries or code modules from other programming languages
Hardware settings

It should be noted that these components are generally consistent across programming languages – they are by no means unique to LabVIEW.
The bottom line is that though the items I listed are all common components, it is up to the developer (or team of developers) to identify the pieces of an application.

We're going to discuss how to manage these larger applications and their various components using the latest tools and recommended practices for use specifically with LabVIEW.
We'll start be examining how to organize application components on disk



Recommendations for Organizing Files

- ***Establish Guidelines Early***
- All files should be stored within a root directory
- Use logical and descriptive naming conventions
- Separate the top level VI from other code
- Group or 'bucket' files according to criteria, such as:
  - Function
  - File type
  - Hierarchical layer

ni.com

45

NATIONAL INSTRUMENTS

What problem is this information trying to solve?
If you're developing an application in LabVIEW, you have multiple components that you need to be able to keep track of
Note: Most people are going to be familiar with the pain of organizing their application, so relate to the problems they've had

The solution presented here
Ideally, you want a set of files that is well organized so that you have a clear understanding of where to find certain components and where to store new components.
This is the guidelines we follow at National Instruments, but how you do things like group files is often up to the developer and depends upon what the application is.

Organizing Files on Disk

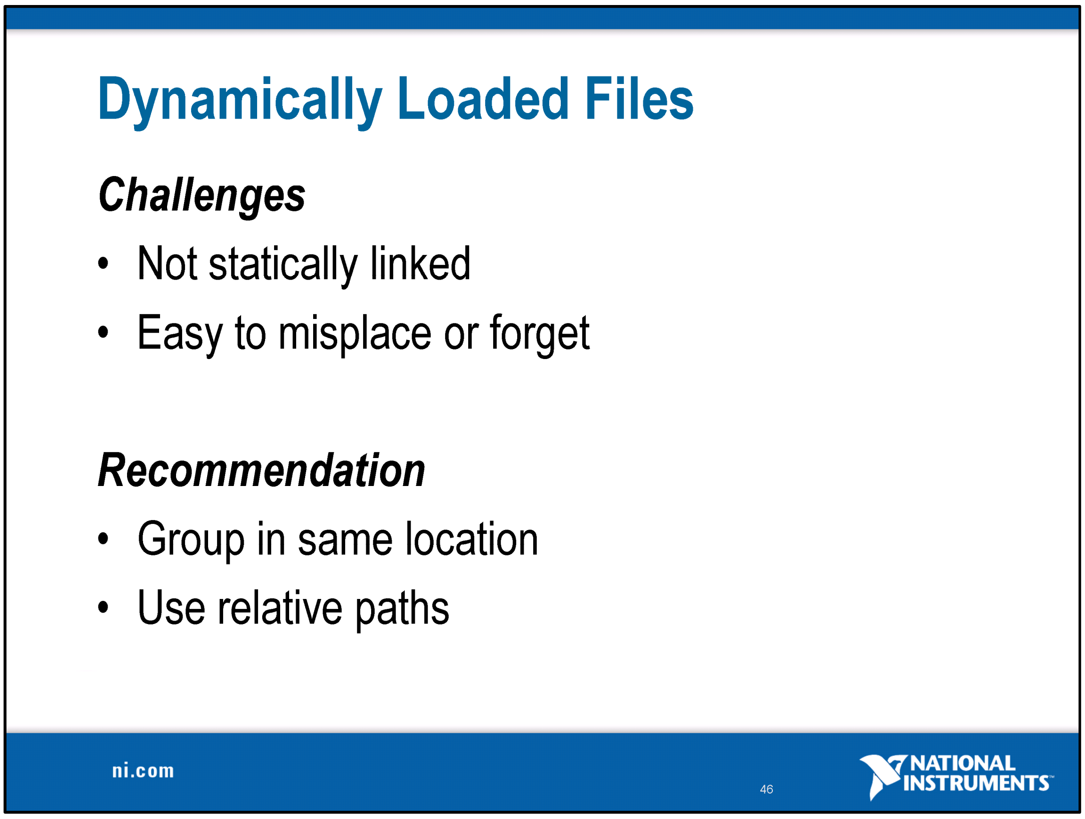
Emphasize the importance of having a logical structure implemented on disk
Discuss criteria (methods) for organizing files on disk
Recommend practices for organization through a demonstration

In this presentation we're going to discuss and explore the practices that we recommend when managing your application; however, many of you may abide by other conventions or practices. We're going to highlight and explain what our developers do internally at National Instruments

Create a directory for all the VIs for one application and give it a meaningful name. Save the main VIs in this directory and the subVIs in a subdirectory. If the subVIs have subVIs, continue the directory hierarchy downward.

Organize the VIs in the file system to reflect the hierarchical nature of the software. Make top-level VIs directly accessible. Place subVIs in subdirectories and group them to reflect any modular components you have designed according to function, such as instrument drivers, configuration utilities, and file I/O drivers.

When naming VIs, VI libraries, and directories, avoid using characters that are not accepted by all file systems, such as slash (/), backslash (\), colon (:), and tilde (~). Avoid creating files with the same name anywhere within the hierarchy. If you have a VI with a specific name in memory and you attempt to load another VI that references a subVI of the same name, the VI links to the VI in memory. If you make backup copies of files, be sure to save them into a directory outside the normal search hierarchy so that LabVIEW does not mistakenly load them into memory when you open development VIs. These are frequent causes of cross-linking. LabVIEW provides tools to resolve these scenarios if they occur, which we will discuss later in this presentation.



Dynamically Loaded Files

Challenges

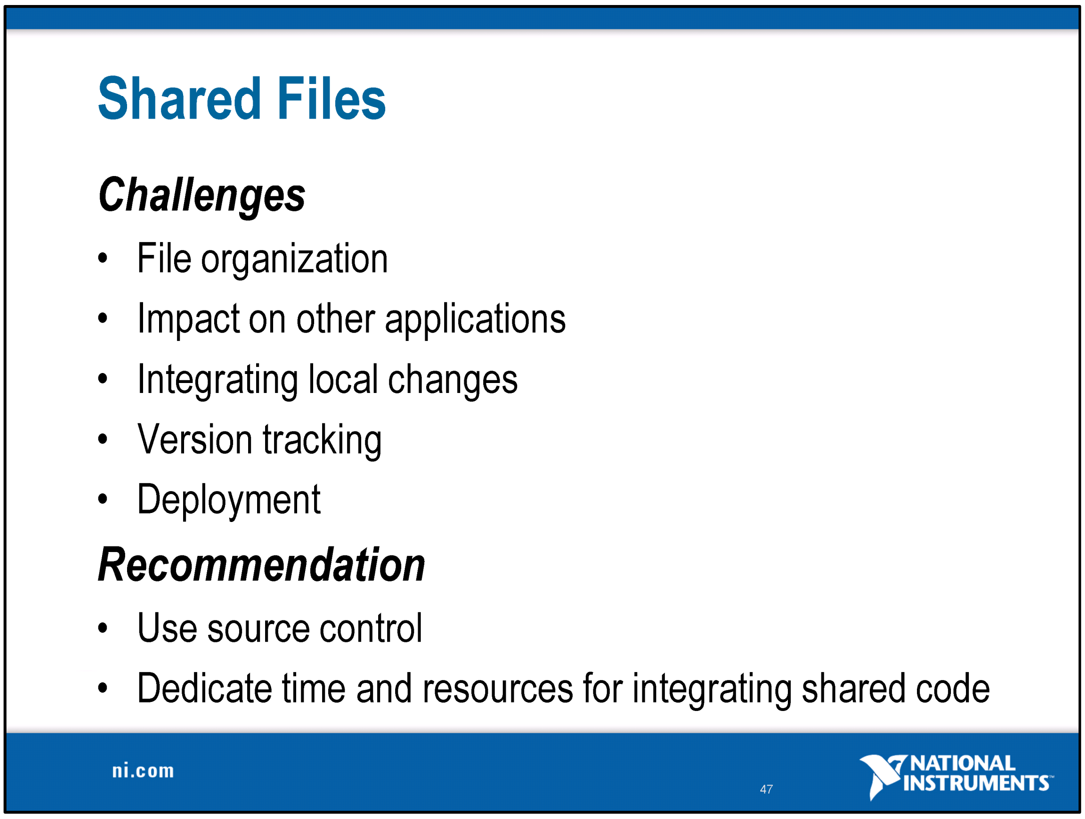
- Not statically linked
- Easy to misplace or forget

Recommendation

- Group in same location
- Use relative paths

ni.com

46



Shared Files

Challenges

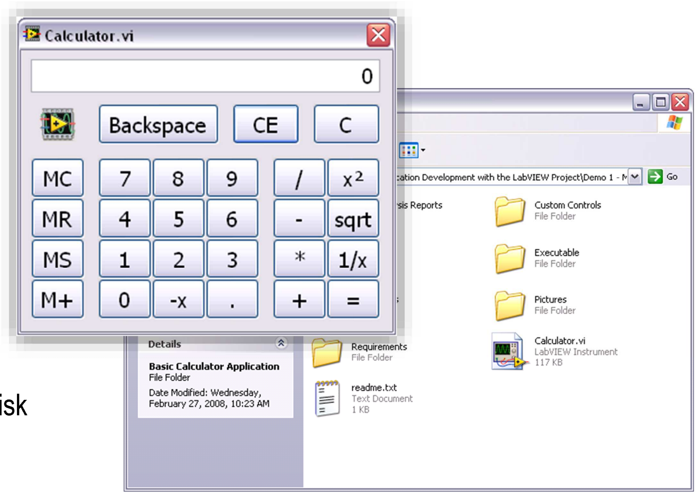
- File organization
- Impact on other applications
- Integrating local changes
- Version tracking
- Deployment

Recommendation

- Use source control
- Dedicate time and resources for integrating shared code

ni.com

47



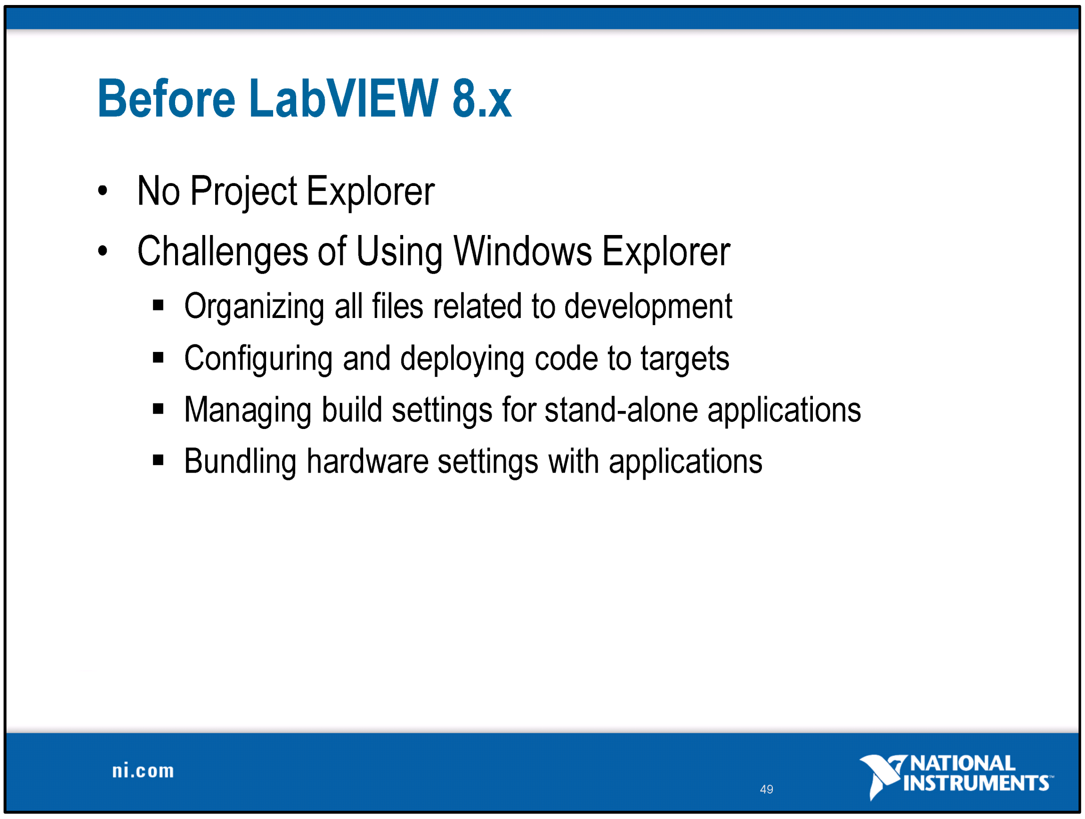
Basic Calculator Application
File Folder
Date Modified: Wednesday, February 27, 2008, 10:23 AM

Organizing Files on Disk

DEMO

Examine an application developed prior to LabVIEW 8.x and organization

Everyone’s used the windows calculator…
This slide is just a chance to introduce the application we’ll be using for the demonstrations in this presentation. It’s a basic four function calculator written in LabVIEW that is very similar to the Windows Calculator.



Before LabVIEW 8.x

- No Project Explorer
- Challenges of Using Windows Explorer
  - Organizing all files related to development
  - Configuring and deploying code to targets
  - Managing build settings for stand-alone applications
  - Bundling hardware settings with applications

ni.com 49 NATIONAL INSTRUMENTS™

Remind developers familiar with LabVIEW that prior to 8.0 the only method for organizing files was through the use of the Operating System file browser (ie: Windows Explorer).

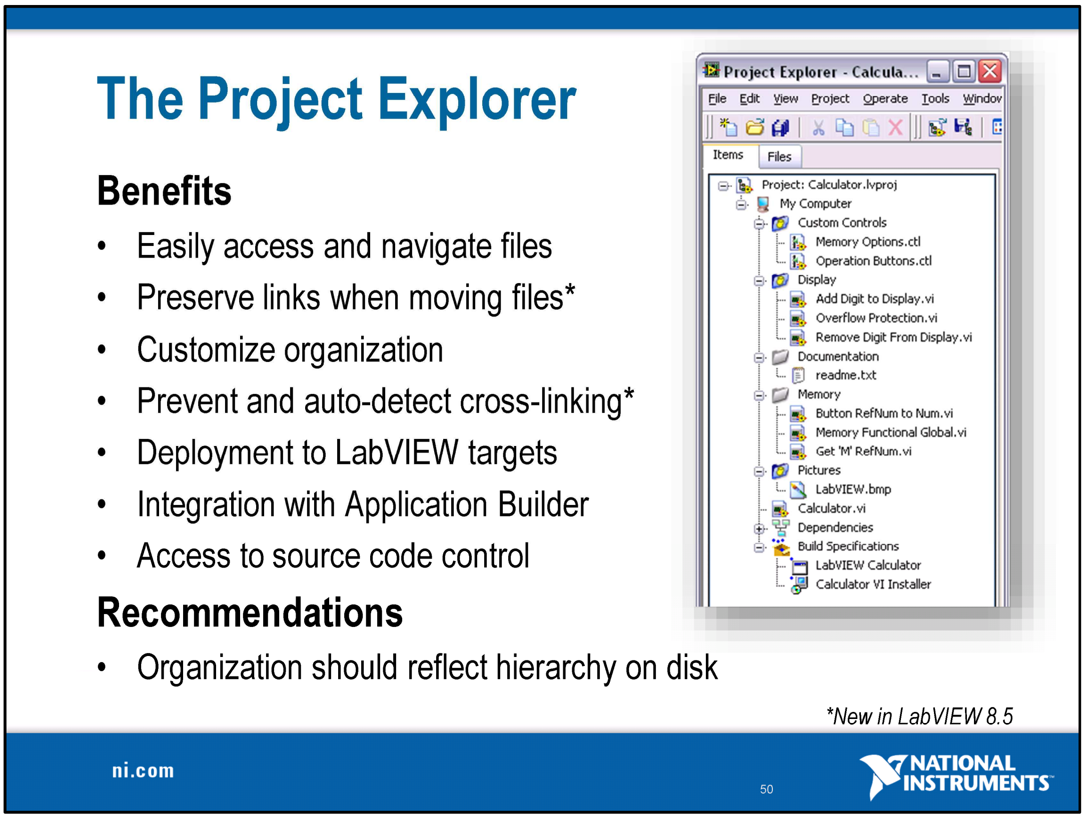
Relate to experience prior to LabVIEW 8

What did you use to manage your application before the Project? All you really had was the operating system file browser, such as Windows Explorer.

Though this application is fairly small, we can already see that we have a fair amount of resources we need to keep track of. Imagine a much larger application or the complexities involved with keeping track of code that is shared across applications or used for group development.

These concerns become compounded if you're trying to manage multiple developers on the same application

LabVIEW 8.0 introduced a new tool that some of you may be familiar with to help address these concerns and help manage and organize application components….



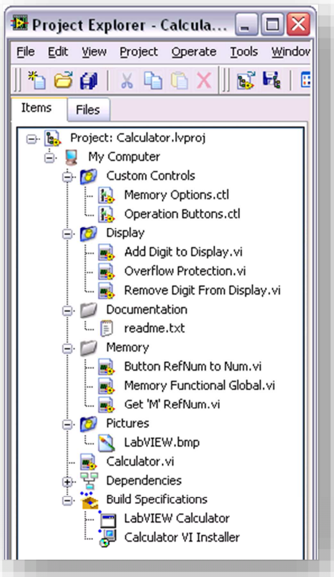
The Project Explorer

Benefits

- Easily access and navigate files
- Preserve links when moving files*
- Customize organization
- Prevent and auto-detect cross-linking*
- Deployment to LabVIEW targets
- Integration with Application Builder
- Access to source code control

Recommendations

- Organization should reflect hierarchy on disk

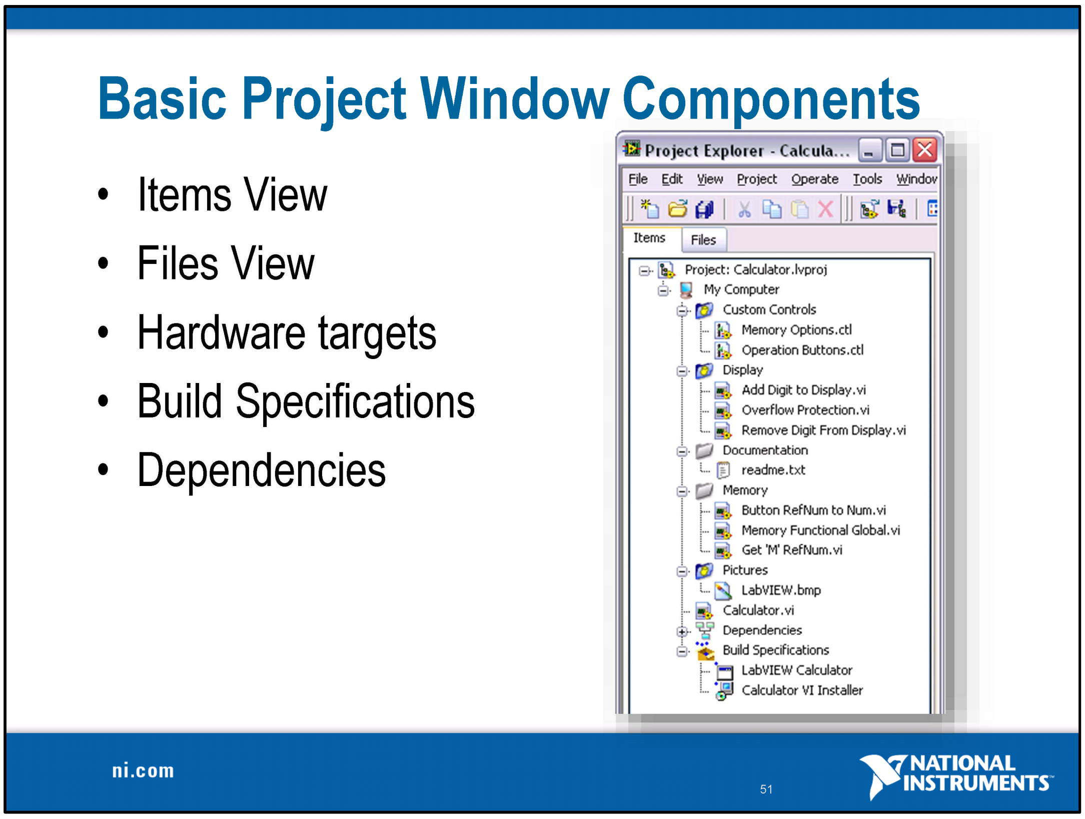

Project Explorer - Calcula...
File Edit View Project Operate Tools Window
Items Files

```
Project: Calculator.lvproj
  My Computer
    Custom Controls
      Memory Options.ctl
      Operation Buttons.ctl
    Display
      Add Digit to Display.vi
      Overflow Protection.vi
      Remove Digit From Display.vi
    Documentation
      readme.txt
    Memory
      Button RefNum to Num.vi
      Memory Functional Global.vi
      Get 'M' RefNum.vi
    Pictures
      LabVIEW.bmp
    Calculator.vi
    Dependencies
    Build Specifications
      LabVIEW Calculator
      Calculator VI Installer
```

**New in LabVIEW 8.5*

ni.com 50 NATIONAL INSTRUMENTS

The project originally came out in LabVIEW 8, but we've made a lot of changes as a result of customer feedback in LabVIEW 8.5
-Customize layout on disk to suit needs
-Should closely resemble hierarchy on disk



Basic Project Window Components

- Items View
- Files View
- Hardware targets
- Build Specifications
- Dependencies

Project Explorer - Calcula...

File Edit View Project Operate Tools Windov

Items | Files

```
Project: Calculator.lvproj
  My Computer
    Custom Controls
      Memory Options.ctl
      Operation Buttons.ctl
    Display
      Add Digit to Display.vi
      Overflow Protection.vi
      Remove Digit From Display.vi
    Documentation
      readme.txt
    Memory
      Button RefNum to Num.vi
      Memory Functional Global.vi
      Get 'M' RefNum.vi
    Pictures
      LabVIEW.bmp
    Calculator.vi
    Dependencies
    Build Specifications
      LabVIEW Calculator
      Calculator VI Installer
```

ni.com

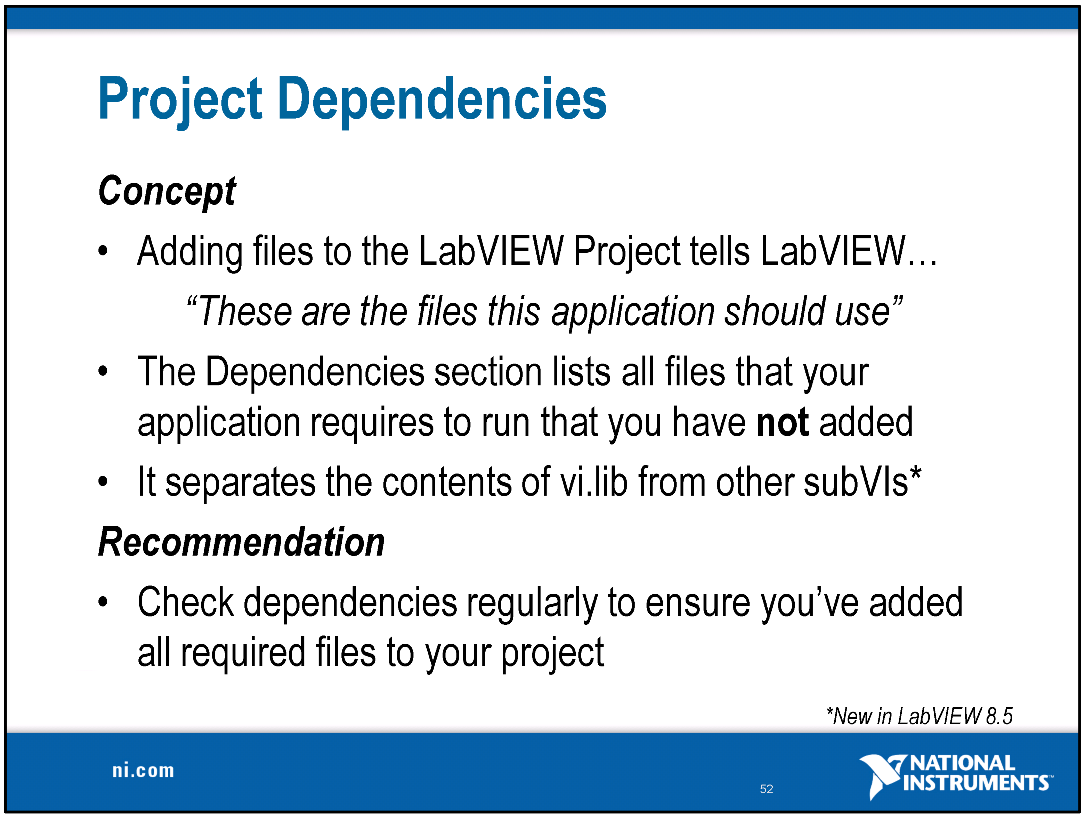
51

NATIONAL INSTRUMENTS

These are the basic components in the Project Explorer

Project file (lvproj) is listed at the top of the Project Explorer
Underneath, we see all hardware targets we're using
Every hardware target has build specifications and dependencies

*Dependencies was changed in LabVIEW 8.5 to separate your files from the contents of vilib



Project Dependencies

Concept

- Adding files to the LabVIEW Project tells LabVIEW…

  “These are the files this application should use”
- The Dependencies section lists all files that your application requires to run that you have **not** added
- It separates the contents of vi.lib from other subVIs*

Recommendation

- Check dependencies regularly to ensure you’ve added all required files to your project

**New in LabVIEW 8.5*

Improved File Management

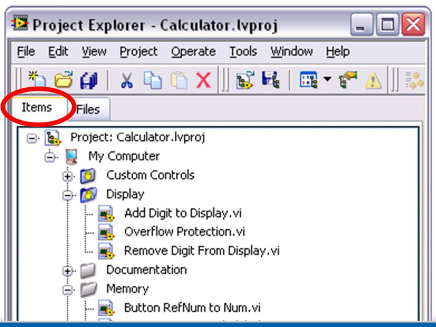
Items View

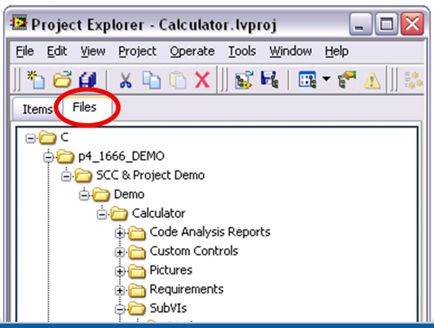
- Customize and synchronize the organization of Project files
- Keep track of project items such as build specifications

Files View*

- Explore filtered disk contents
- Safely move and rename files on disk without breaking VI links

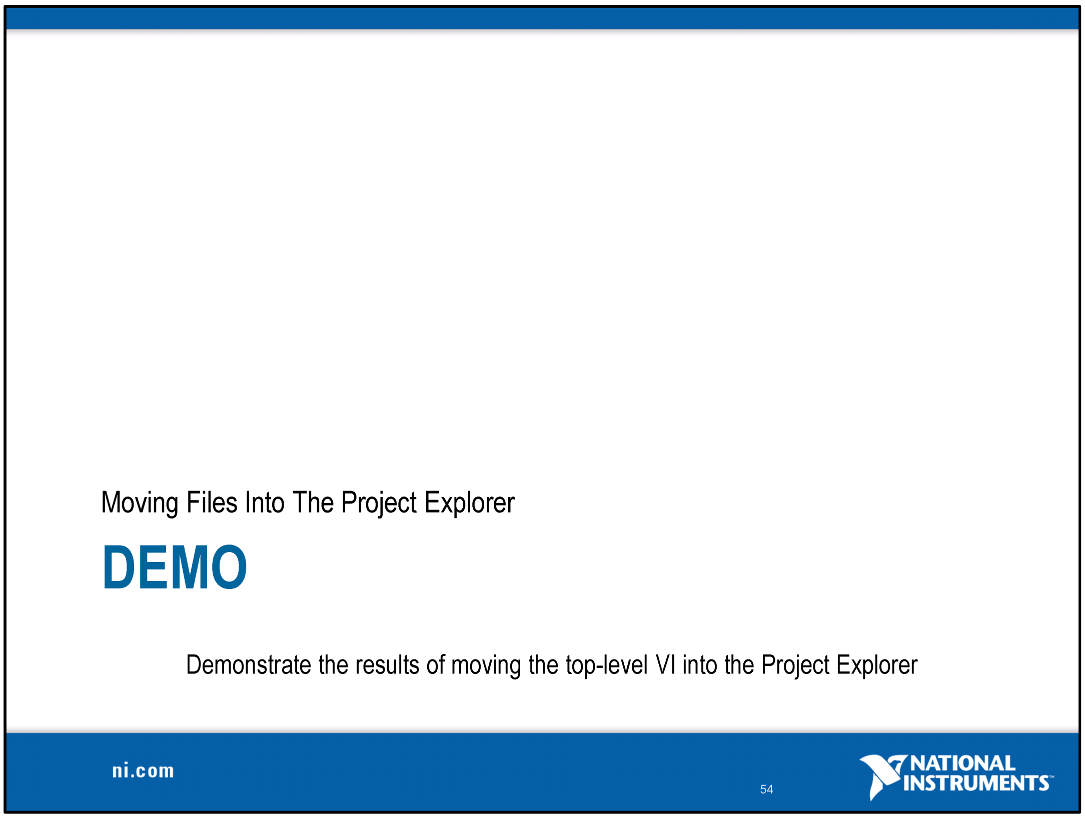
**New in LabVIEW 8.5*


Project Explorer - Calculator.lvproj
File Edit View Project Operate Tools Window Help
Items
Files
Project: Calculator.lvproj
My Computer
Custom Controls
Display
Add Digit to Display.vi
Overflow Protection.vi
Remove Digit From Display.vi
Documentation
Memory
Button RefNum to Num.vi



The Files View is also new in LabVIEW 8.5. This is an important feature for organizing files on disk and for performing file operations. Copy, rename and delete files from this view in order to avoid breaking links between files

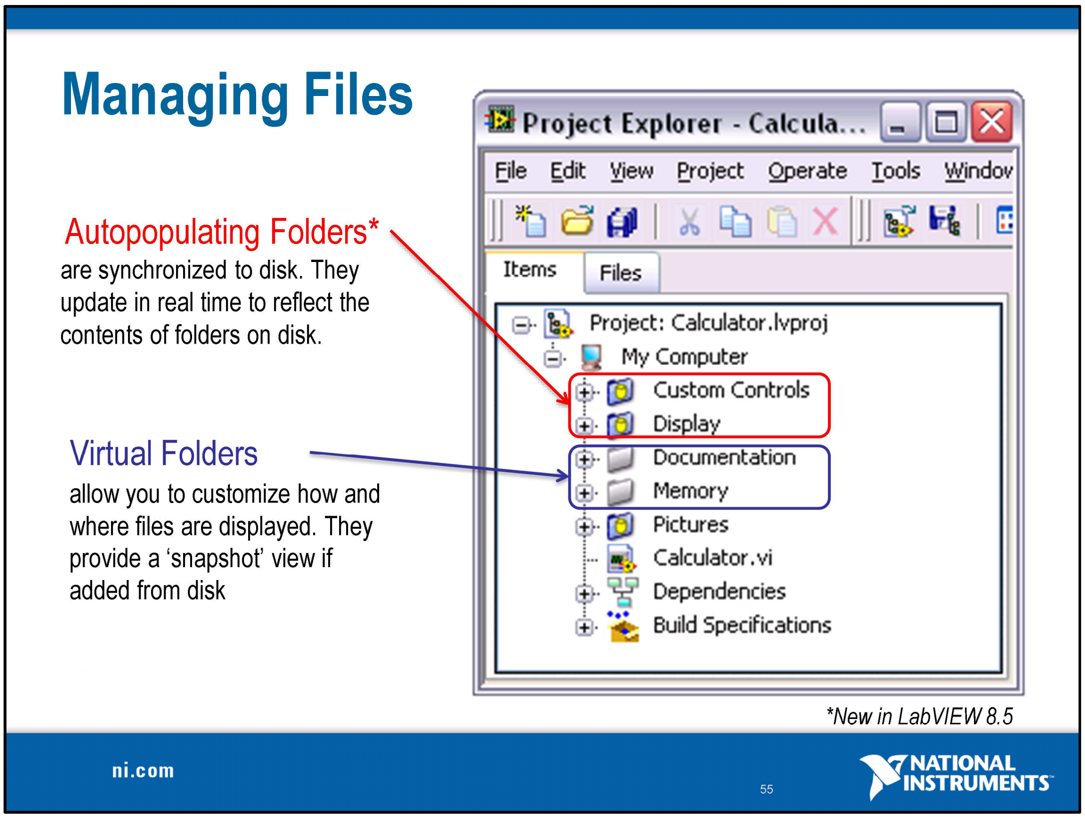
Files view provides a hierarchy similar to what you will see in the Project Explorer; however, it filters out files that are not in your project



Moving Files Into The Project Explorer

DEMO

Demonstrate the results of moving the top-level VI into the Project Explorer



Managing Files

Autopopulating Folders* are synchronized to disk. They update in real time to reflect the contents of folders on disk.

Virtual Folders allow you to customize how and where files are displayed. They provide a 'snapshot' view if added from disk

*New in LabVIEW 8.5

ni.com 55 NATIONAL INSTRUMENTS

We use folders to organize and bucket files in the Project Explorer.
LabVIEW 8.5 introduces a new folder type, called Autopopulating folders

The files are readily available and that you can get at them and access them easily. In addition, use of the project provides you flexibility that you don't have otherwise.

Even though it's more important to talk about these folders while actually using this project, this slide may prove useful in showing the distinction because it's an enlarged screenshot (these icons may be hard to see for your audience otherwise)

Auto-Populating Folders

For most users, it is desirable that the organization of source files within the Project reflect the organization of files on disk. To accomplish this, you can use auto-populating Project folders. Auto-populating folders synchronize their contents within LabVIEW to the contents to specified directories on disk, ensuring that modifications to the hierarchy need only be made once in order to appear in both locations.

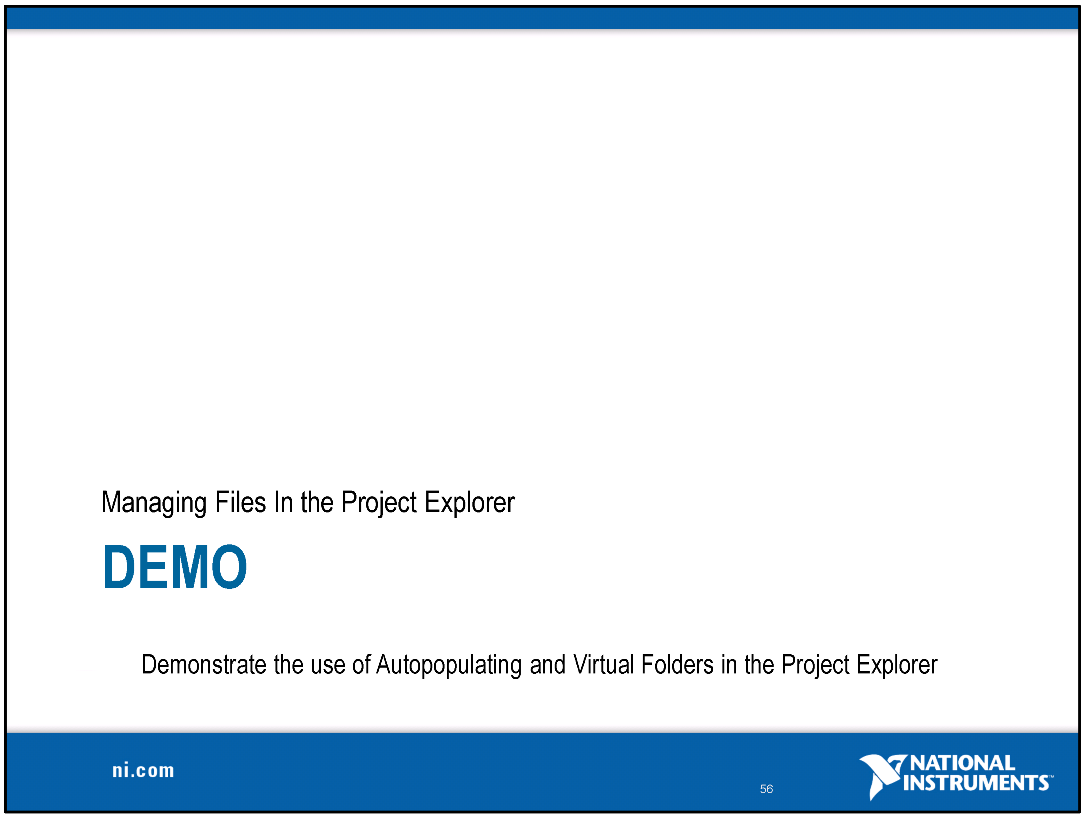
If a project library is stored within a directory that is synchronized to an auto-populating folder the project library and the files it refers to will appear in the Project Explorer. This is not recommended since these libraries can have implicit representations to files and folders that are not within the same folder.

Virtual Folders

Virtual folders are logical folders that make it possible for you to display and organize files in the Project Explorer without regard for organization on disk; virtual folders do not necessarily reflect the organization of file directories on disk. You can import a folder from disk into the Project as a virtual folder, but it will effectively be a snapshot of the folder at the time you imported it.

Virtual folders enable you to completely customize the organization of files within the Items view of the Project Explorer without rearranging files on disk, which avoids disrupting other applications that share the code or creating cross-linked files.

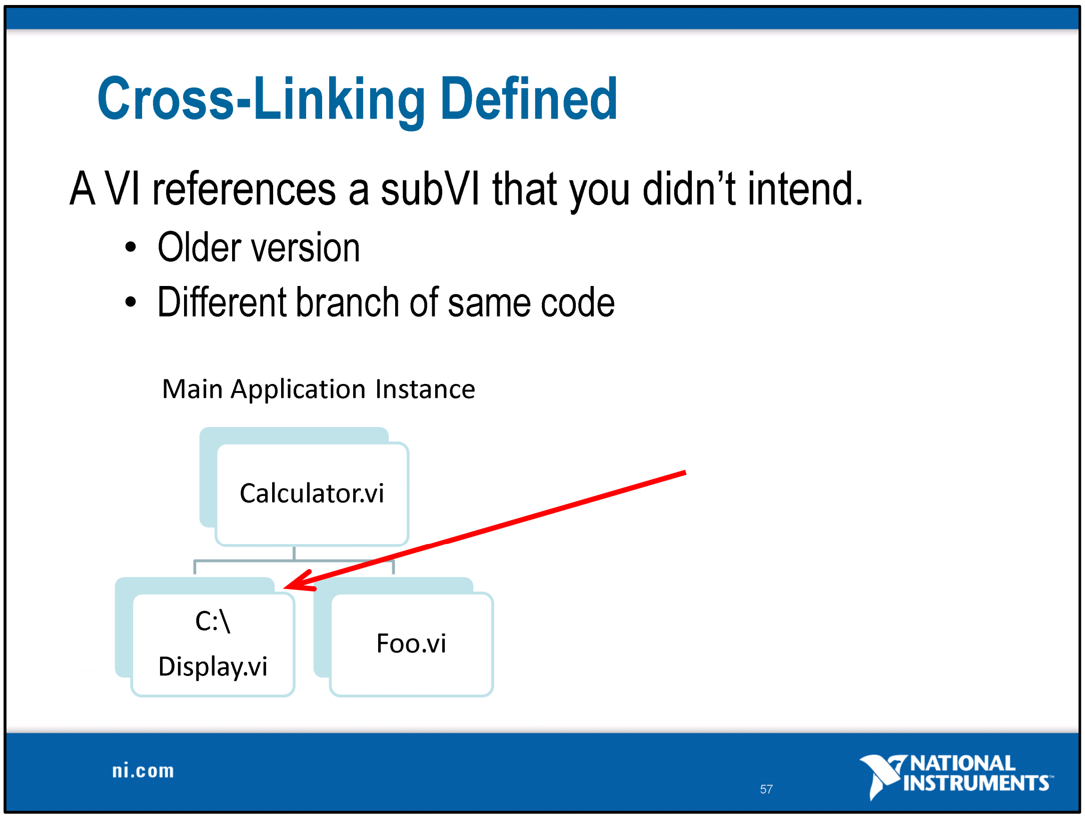
However, note that if the organization of files within the Project Explorer differs greatly from the physical layout on disk, you may have a difficult time updating and maintaining the application.



Managing Files In the Project Explorer

DEMO

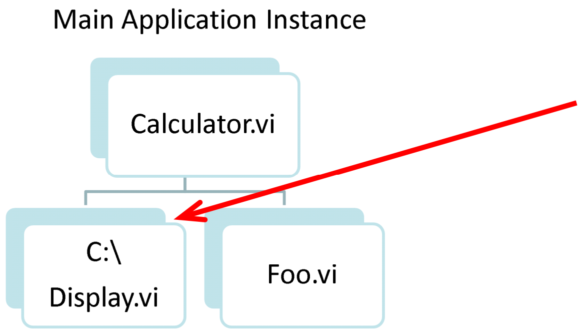
Demonstrate the use of Autopopulating and Virtual Folders in the Project Explorer



Cross-Linking Defined

A VI references a subVI that you didn't intend.

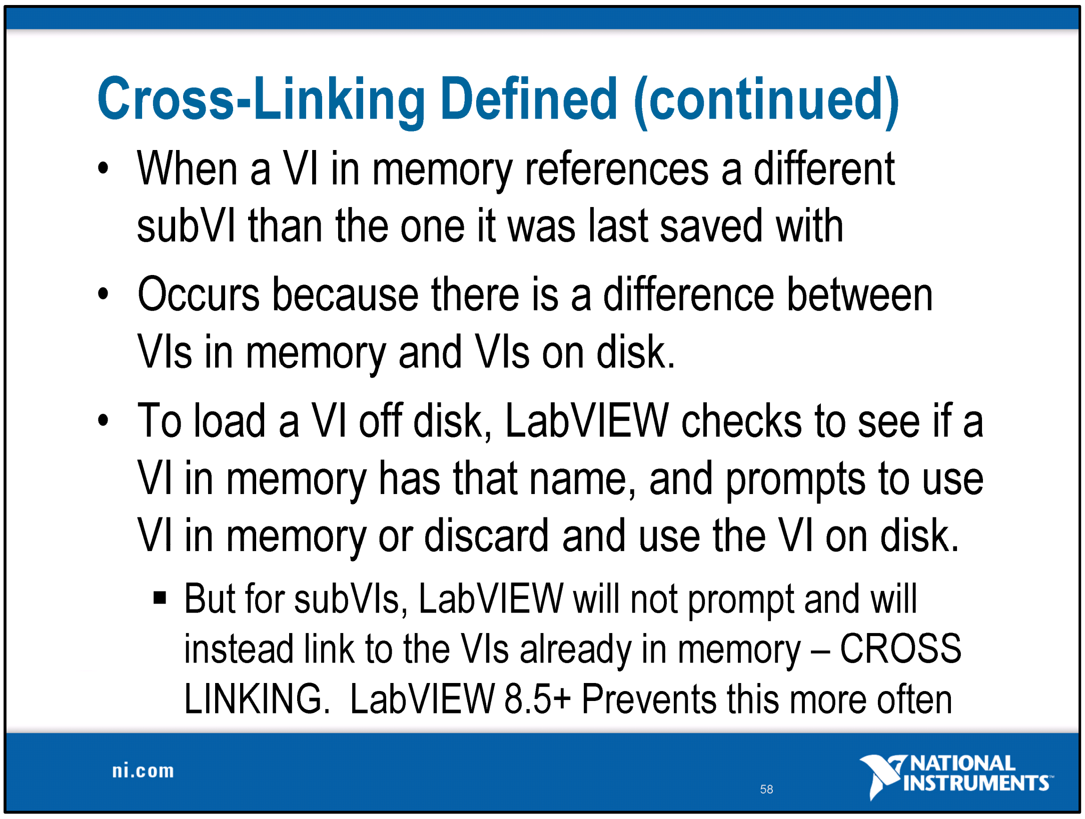
- Older version
- Different branch of same code



Cross-linking is when a VI in memory references a different subVI than the one it was last saved with, one that the developer didn't intend to reference. Cross-linking occurs because there is a difference between VIs in memory and VIs on disk. Items in memory are unique by name, but multiple VIs of the same name can exist on disk.

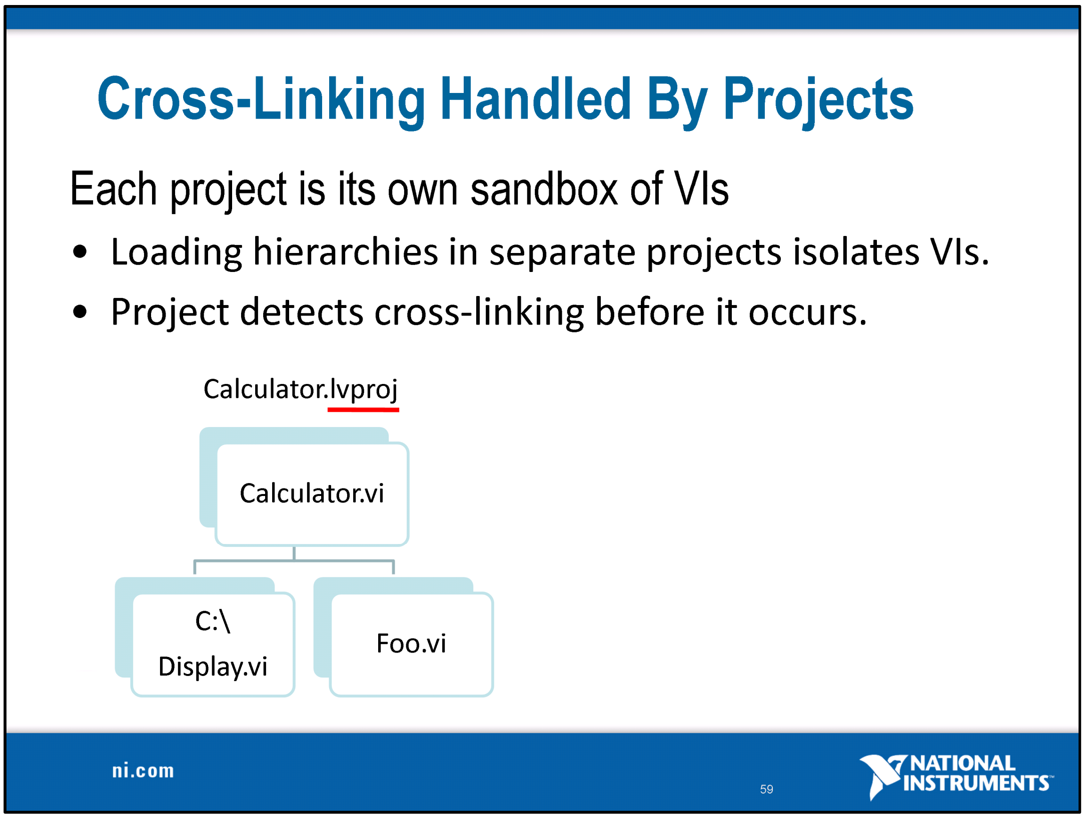
Therefore, when you try to load a VI off disk, LabVIEW first checks to see if a VI in memory has that name, and prompts you to use the one in memory or Discard it and use the one from disk. But for subVIs, LabVIEW will not prompt and will instead link to the VIs already in memory and present a Warnings dialog informing you that it loaded VIs from unexpected locations.

Cross-linking is when a VI in memory references a different subVI than the one it was last saved with, one that the developer didn't intend to reference.
Cross-linking occurs because there is a difference between VIs in memory and VIs on disk.
Items in memory are unique by name, but multiple VIs of the same name can exist on disk.
Therefore, when you try to load a VI off disk, LabVIEW first checks to see if a VI in memory has that name, and prompts you to use the one in memory or Discard it and use the one from disk.
But for subVIs, LabVIEW will not prompt and will instead link to the VIs already in memory and present a Warnings dialog informing you that it loaded VIs from unexpected locations.



Cross-Linking Defined (continued)

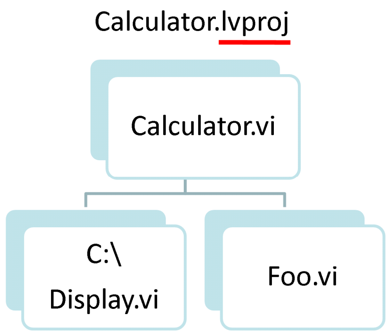
- When a VI in memory references a different subVI than the one it was last saved with
- Occurs because there is a difference between VIs in memory and VIs on disk.
- To load a VI off disk, LabVIEW checks to see if a VI in memory has that name, and prompts to use VI in memory or discard and use the VI on disk.
  - But for subVIs, LabVIEW will not prompt and will instead link to the VIs already in memory – CROSS LINKING. LabVIEW 8.5+ Prevents this more often



Cross-Linking Handled By Projects

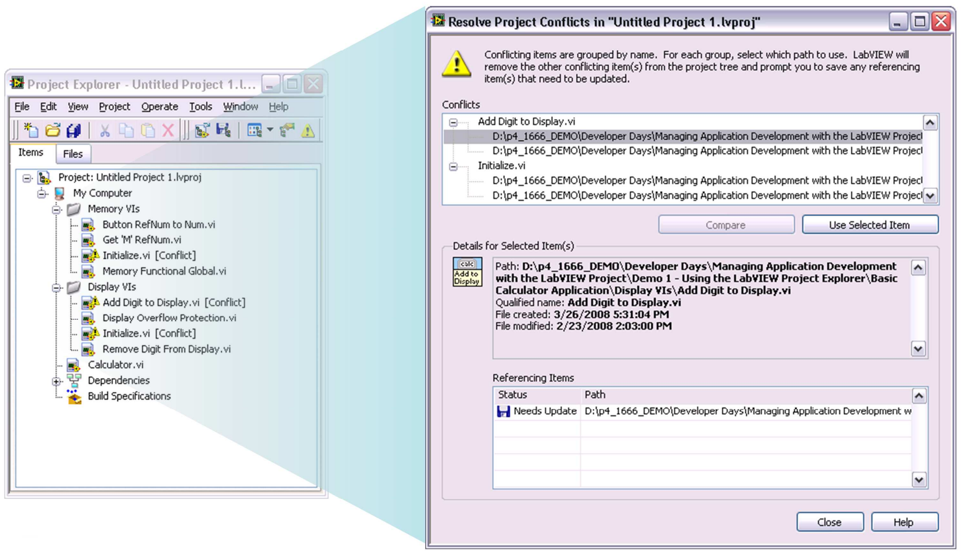
Each project is its own sandbox of VIs

- Loading hierarchies in separate projects isolates VIs.
- Project detects cross-linking before it occurs.



ni.com 59 NATIONAL INSTRUMENTS

Auto-Resolve Cross Linking*


Project Explorer - Untitled Project 1.l...
File Edit View Project Operate Tools Window Help
Items
Files

Project: Untitled Project 1.lvproj
My Computer
Memory VIs
Button RefNum to Num.vi
Get 'M' RefNum.vi
Initialize.vi [Conflict]
Memory Functional Global.vi
Display VIs
Add Digit to Display.vi [Conflict]
Display Overflow Protection.vi
Initialize.vi [Conflict]
Remove Digit From Display.vi
Calculator.vi
Dependencies
Build Specifications

Resolve Project Conflicts in "Untitled Project 1.lvproj"

Conflicting items are grouped by name. For each group, select which path to use. LabVIEW will remove the other conflicting item(s) from the project tree and prompt you to save any referencing item(s) that need to be updated.

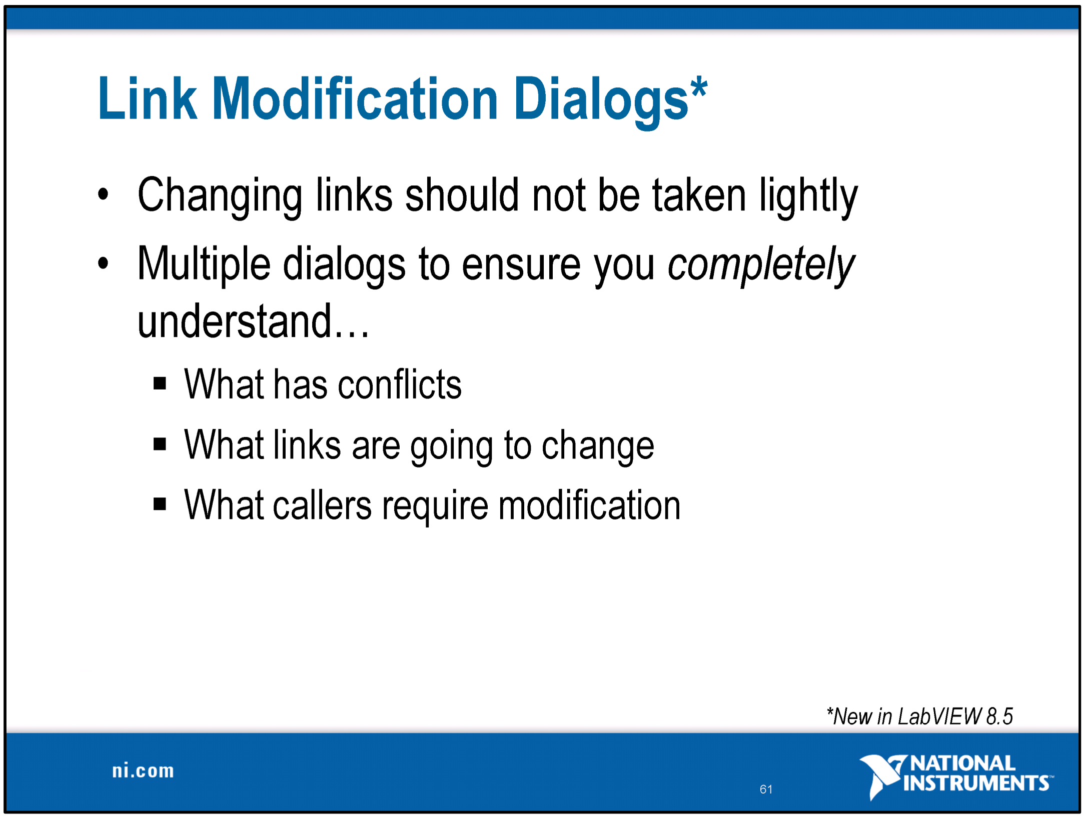
Conflicts
Add Digit to Display.vi
D:\p4_1666_DEMO\Developer Days\Managing Application Development with the LabVIEW Project
D:\p4_1666_DEMO\Developer Days\Managing Application Development with the LabVIEW Project
Initialize.vi
D:\p4_1666_DEMO\Developer Days\Managing Application Development with the LabVIEW Project
D:\p4_1666_DEMO\Developer Days\Managing Application Development with the LabVIEW Project

Details for Selected Item(s)
Path: **D:\p4_1666_DEMO\Developer Days\Managing Application Development with the LabVIEW Project\Demo 1 - Using the LabVIEW Project Explorer\Basic Calculator Application\Display VIs\Add Digit to Display.vi**
Qualified name: **Add Digit to Display.vi**
File created: **3/26/2008 5:31:04 PM**
File modified: **2/23/2008 2:03:00 PM**

Referencing Items

Status	Path
Needs Update	D:\p4_1666_DEMO\Developer Days\Managing Application Development w

**New in LabVIEW 8.5*



Link Modification Dialogs*

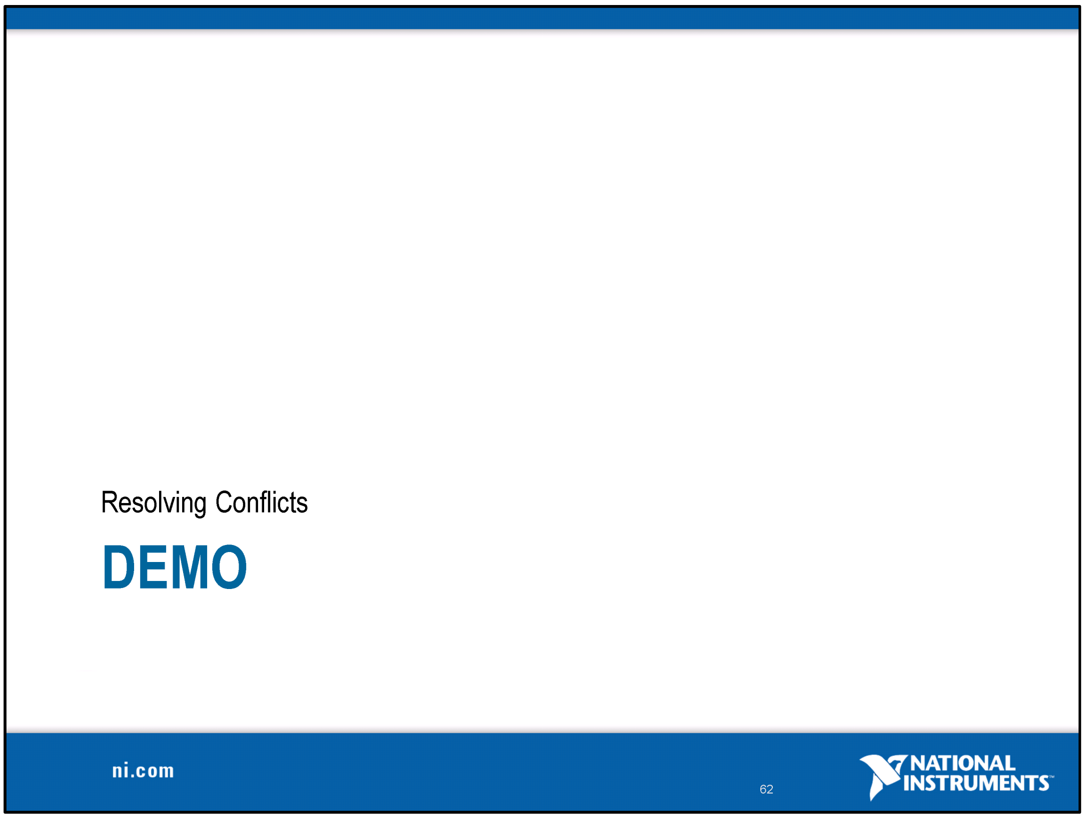
- Changing links should not be taken lightly
- Multiple dialogs to ensure you *completely* understand…
  - What has conflicts
  - What links are going to change
  - What callers require modification

**New in LabVIEW 8.5*

ni.com

61

NATIONAL INSTRUMENTS™

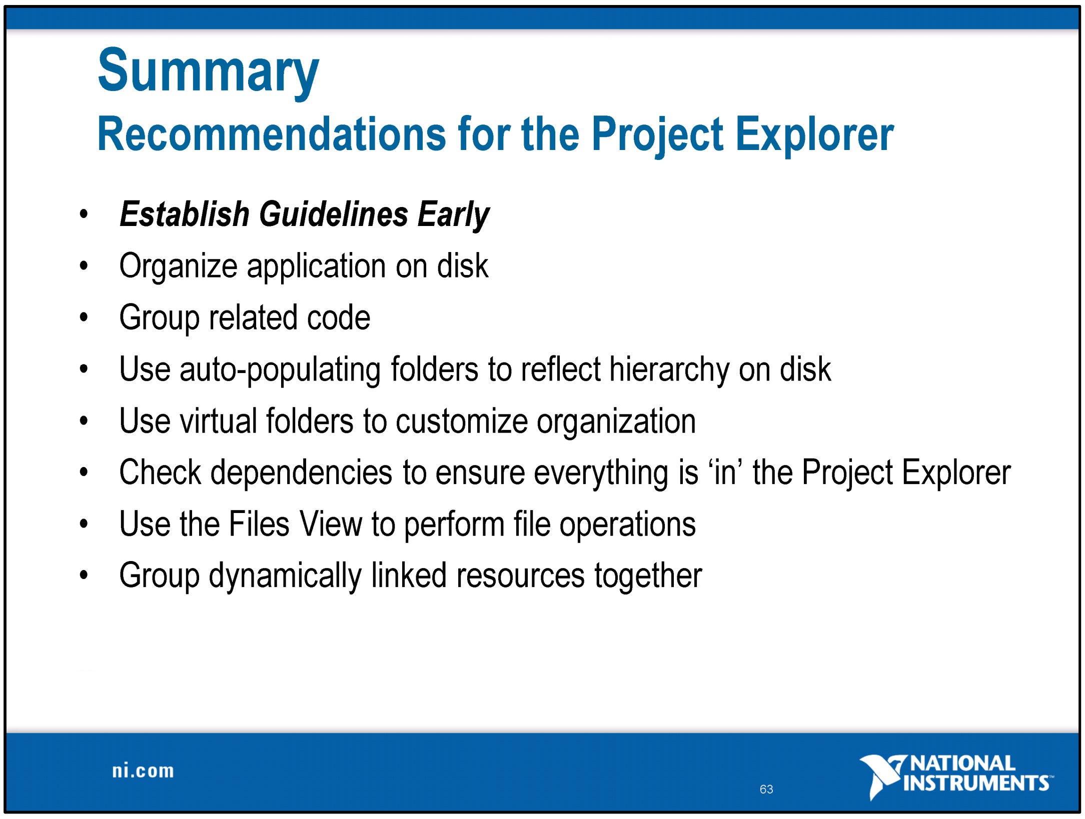


Resolving Conflicts

DEMO

ni.com

62



Summary

Recommendations for the Project Explorer

- ***Establish Guidelines Early***
- Organize application on disk
- Group related code
- Use auto-populating folders to reflect hierarchy on disk
- Use virtual folders to customize organization
- Check dependencies to ensure everything is 'in' the Project Explorer
- Use the Files View to perform file operations
- Group dynamically linked resources together