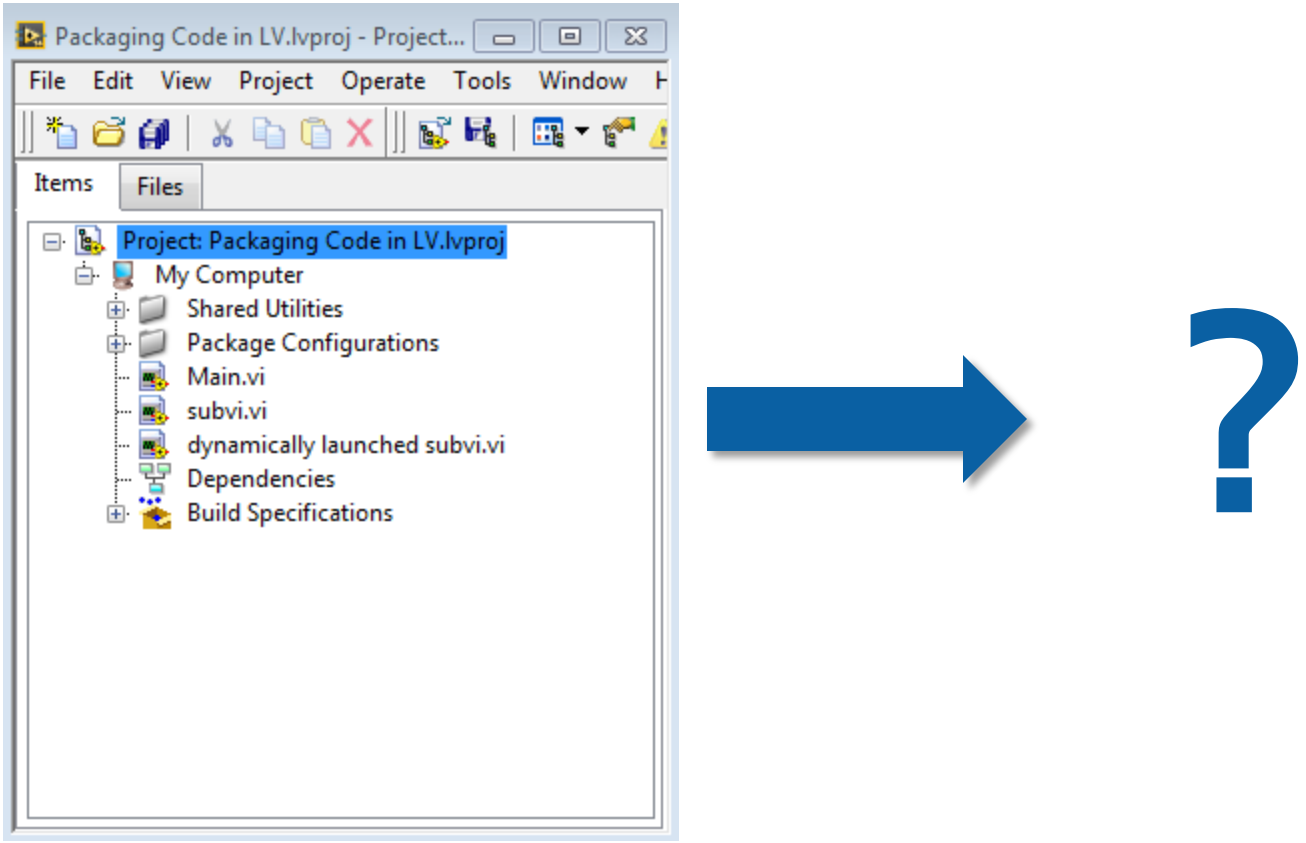
LabVIEW Developer Days

Build Code. Form Communities. Gain Confidence.

NATIONAL INSTRUMENTS™

Best Practices for Code Packaging in LabVIEW

Great LabVIEW Code – Now What?



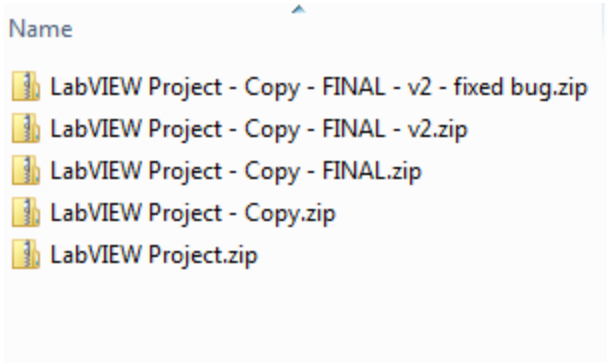
NATIONAL INSTRUMENTS

Four main use cases

- Sharing source code
- Distributing an application
- Sharing a reusable component (API)
- Adding a plug-in to an application

Sharing Source Code

- Another developer needs a copy of your source code
- Naïve approach:
  - Copy folder on disk, make zip file, email to colleague
- Issues:
  - VI dependencies
  - Multiple editors



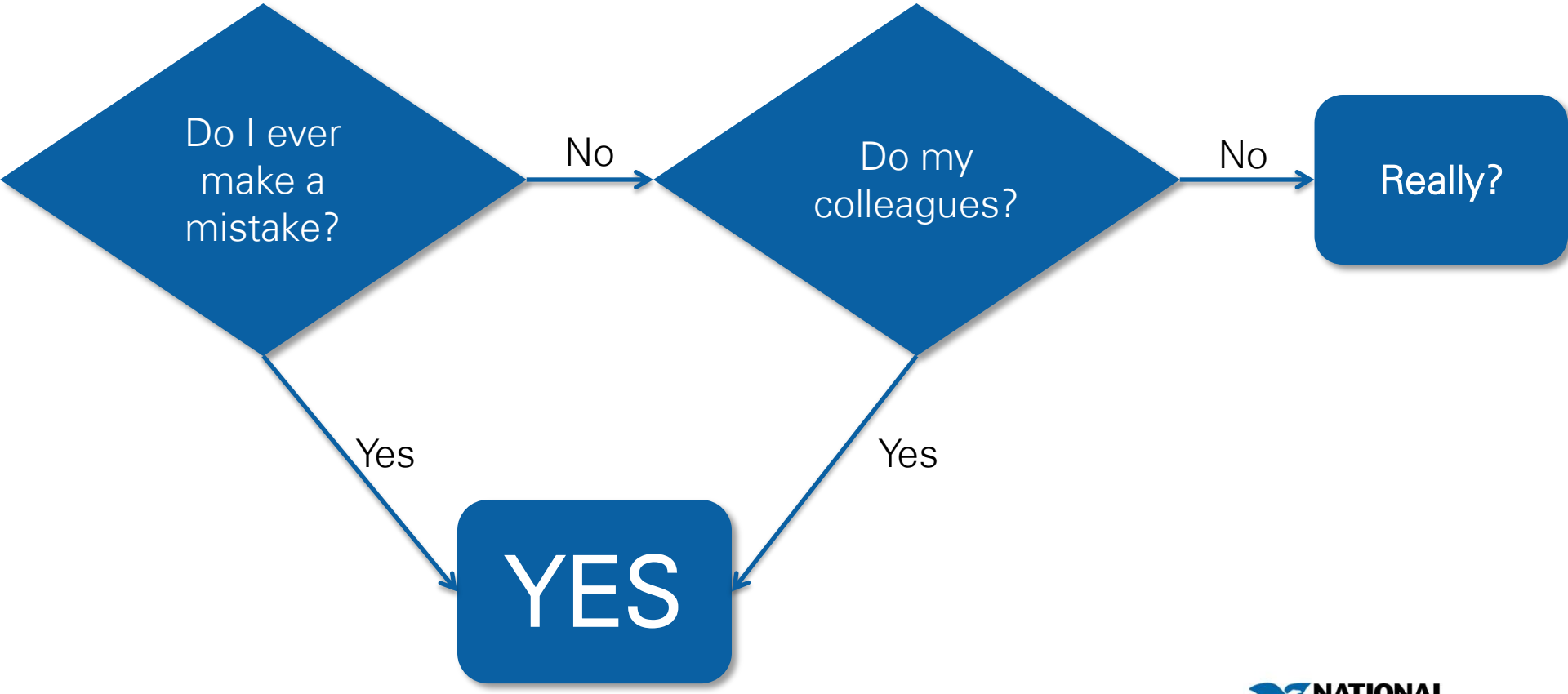
BAD

NATIONAL INSTRUMENTS

Sharing Source Code - What

- Source Code Control
  - Sharing code amongst developers of the same codebase
- Source Distribution
  - Exporting a copy of the code with all dependencies
    - Typically to a developer in a different group or for a different project
  - One option for distributing plug-ins (more on this later)

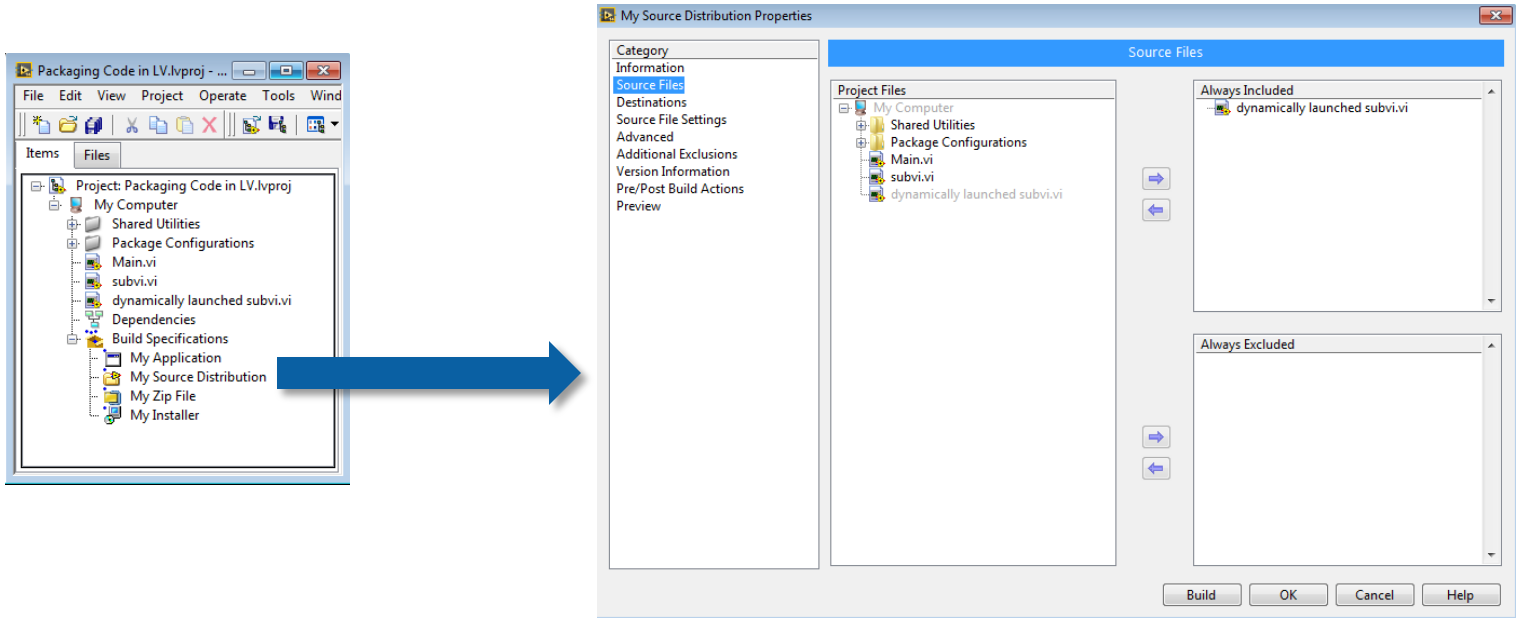
Source Code Control – Do I Need It?



NATIONAL INSTRUMENTS

Sharing Source Code – Source Distributions

- Create a Source Distribution Build Specification
- Select the source files you need
- All dependencies will be automatically included
- Source distributions do not include the project file itself
  - You must manually include the project file in the project



NATIONAL INSTRUMENTS

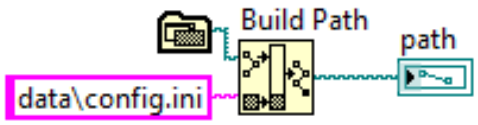
Sharing Applications

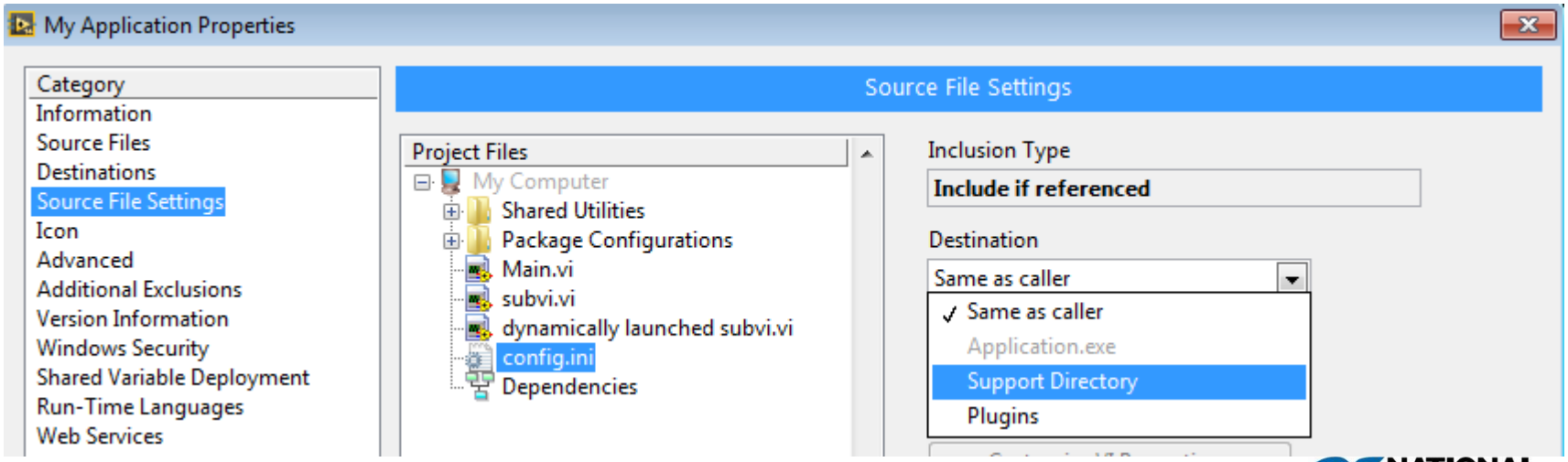
- Executable for standalone applications
- Shared library (DLL/.so/framework) for integration into other environments

- Common pitfalls:
  - Path changes from dev to executable
  - Different behaviors needed in executable vs dev environment

Applications - Paths

- Application Directory Constant
  - (File I/O>>File Constants palette)
  - Returns the project directory when run from dev environment
  - Returns the executable directory when run from executable
- Keep same relative paths in project and executable

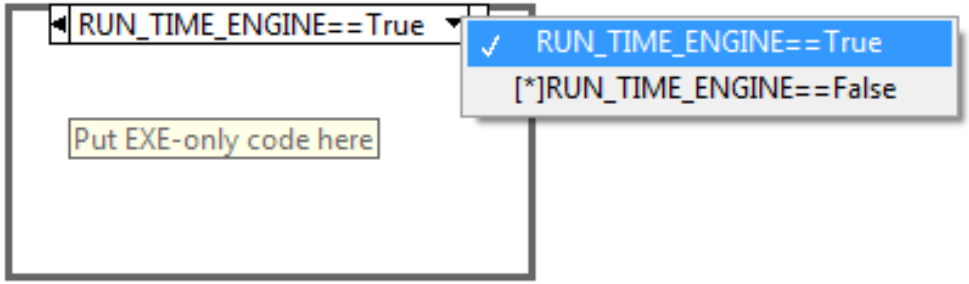




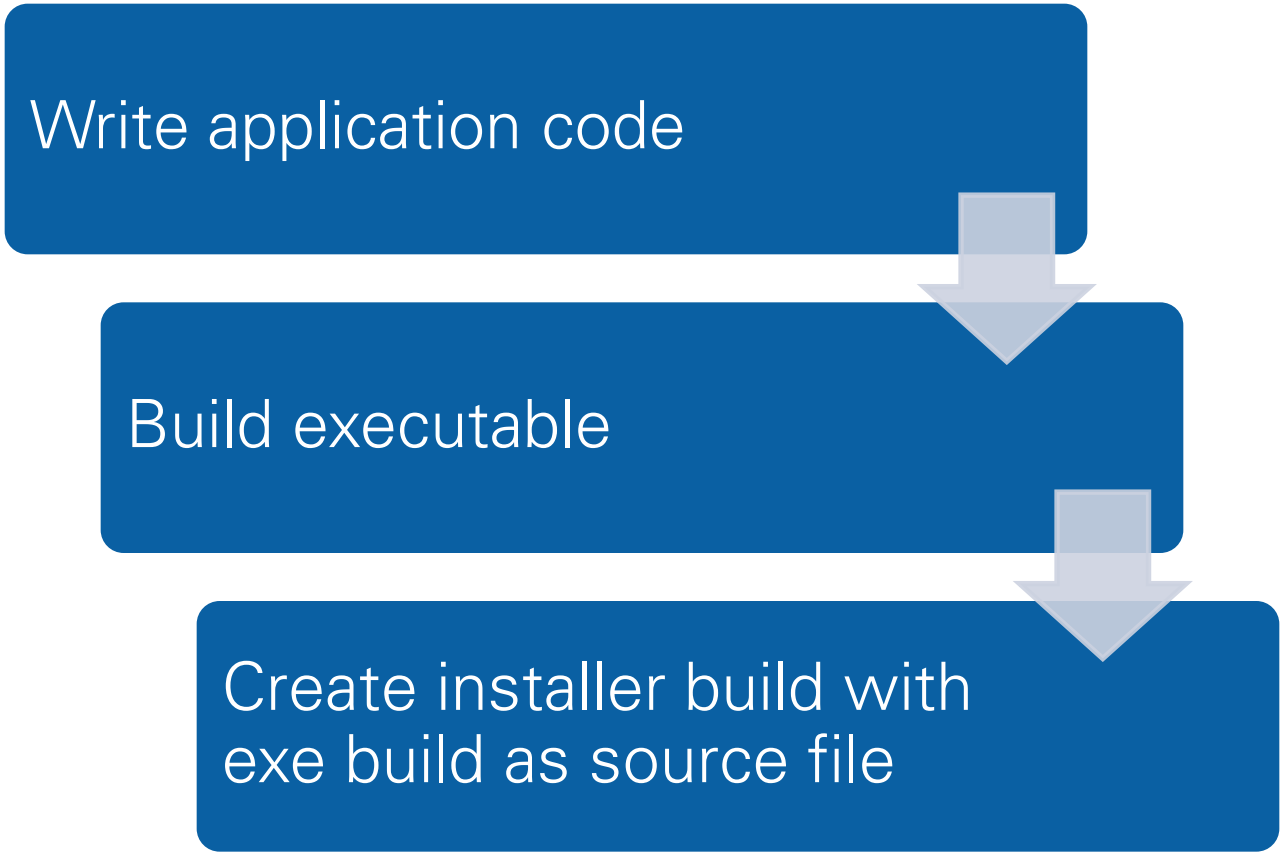
NATIONAL INSTRUMENTS

Applications – Conditional Disable

- Use the Conditional Disable Structure to encapsulate code meant to run only in development environment or as an executable



Sharing Applications – Build Process



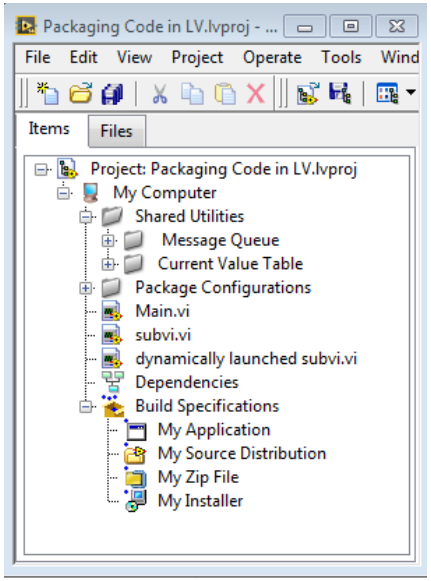
Application - DEMO

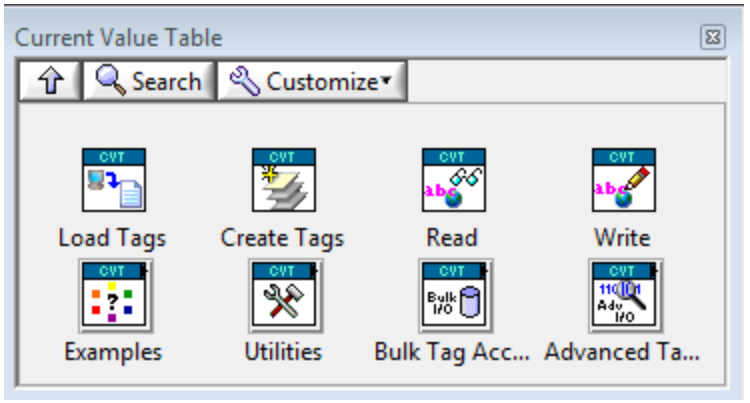
Components

- Naïve approach: make a source distribution and ask a user to put everything in the right place on disk. Or worse, just make a copy for every project.
- Issues:
  - Adding items to the palettes
  - Recompiling for different LabVIEW versions
  - Removing/updating components
  - Dependent components

Components

- Simply copying a ‘shared’ folder is inefficient
  - Hard to update across multiple applications
- Instead, build an API and distribute a component

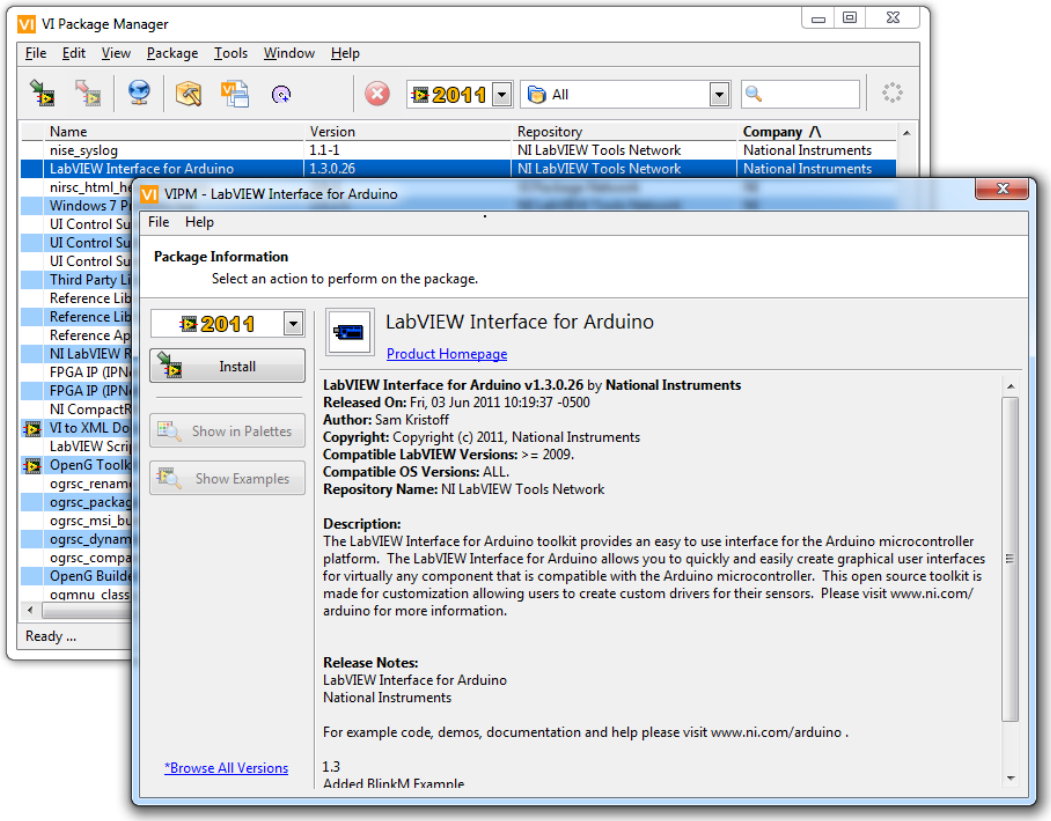




NATIONAL INSTRUMENTS

VI Package Manager for API Distribution

VI Package Manager™



Create
"VI Package" files from LabVIEW code

Find and Install
VI Packages, downloadable from the internet

Manage
VI Packages used in multiple applications

Buy and Sell
VI Packages on the LabVIEW Tools Network, a marketplace on the web.

LabVIEW TOOLS NETWORK

NATIONAL INSTRUMENTS

Components - Locations

vi.lib

- Public reuse components
- /vi.lib/company/component

instr.lib

- Instrument drivers
- /inst.lib/company/instrument

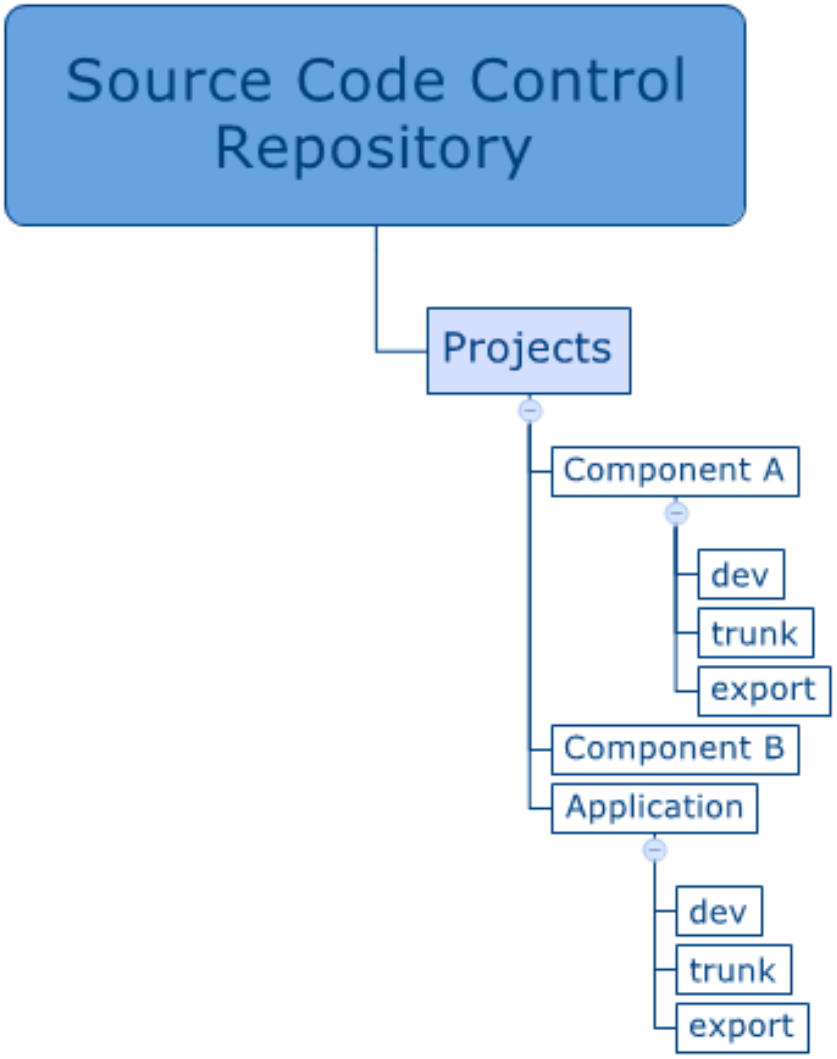
user.lib

- For a user's personal libraries

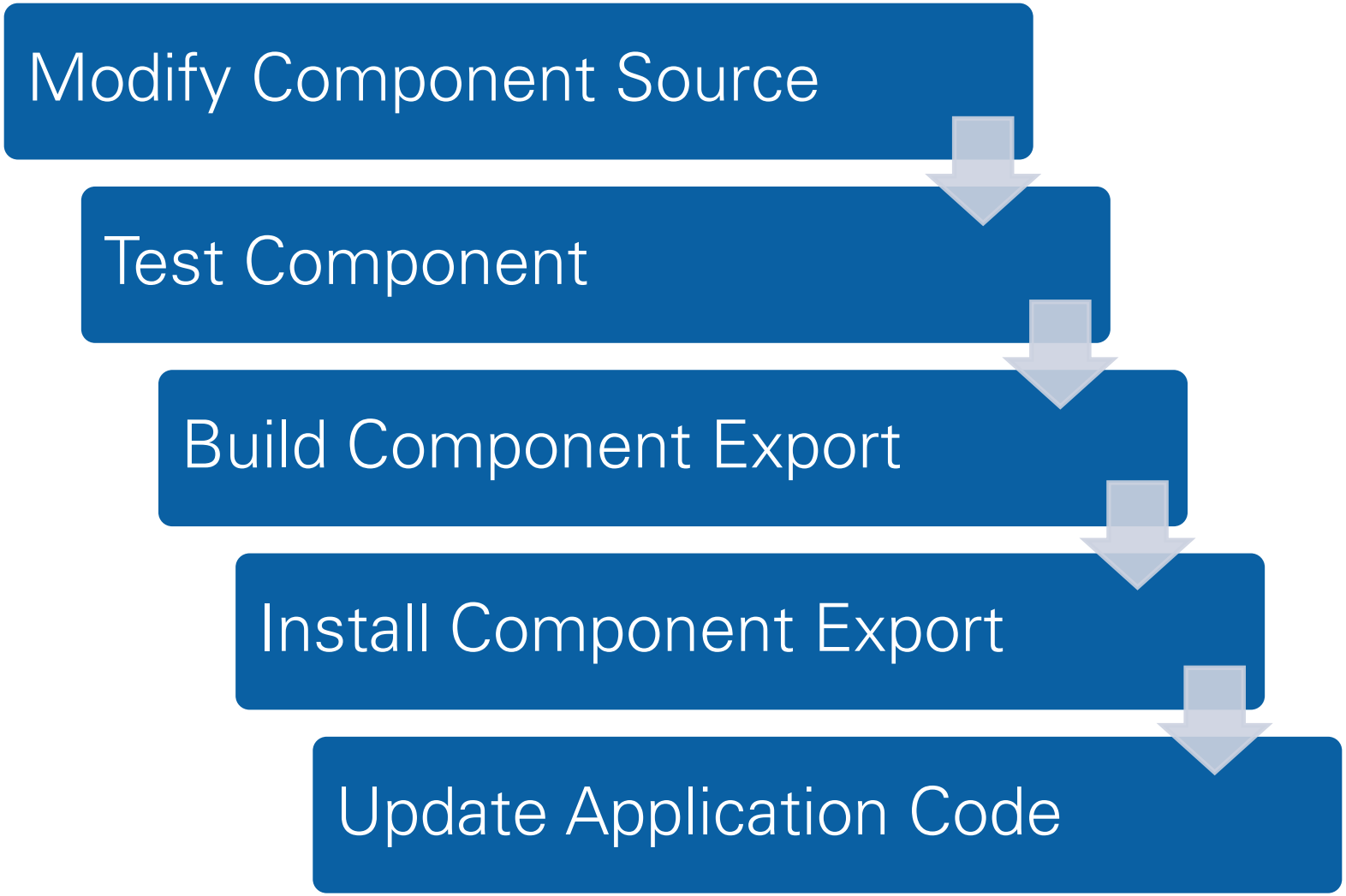
Components – Tips

- Develop your component in a separate project from any application that uses the component
- Never link directly to the source code of a component – always use the export (VI Package, installer, etc)
- Limit dependencies on other components to the minimum necessary
- Avoid making changes to the public API
  - Maintain backward compatibility

Source Code Control – Basic Structure

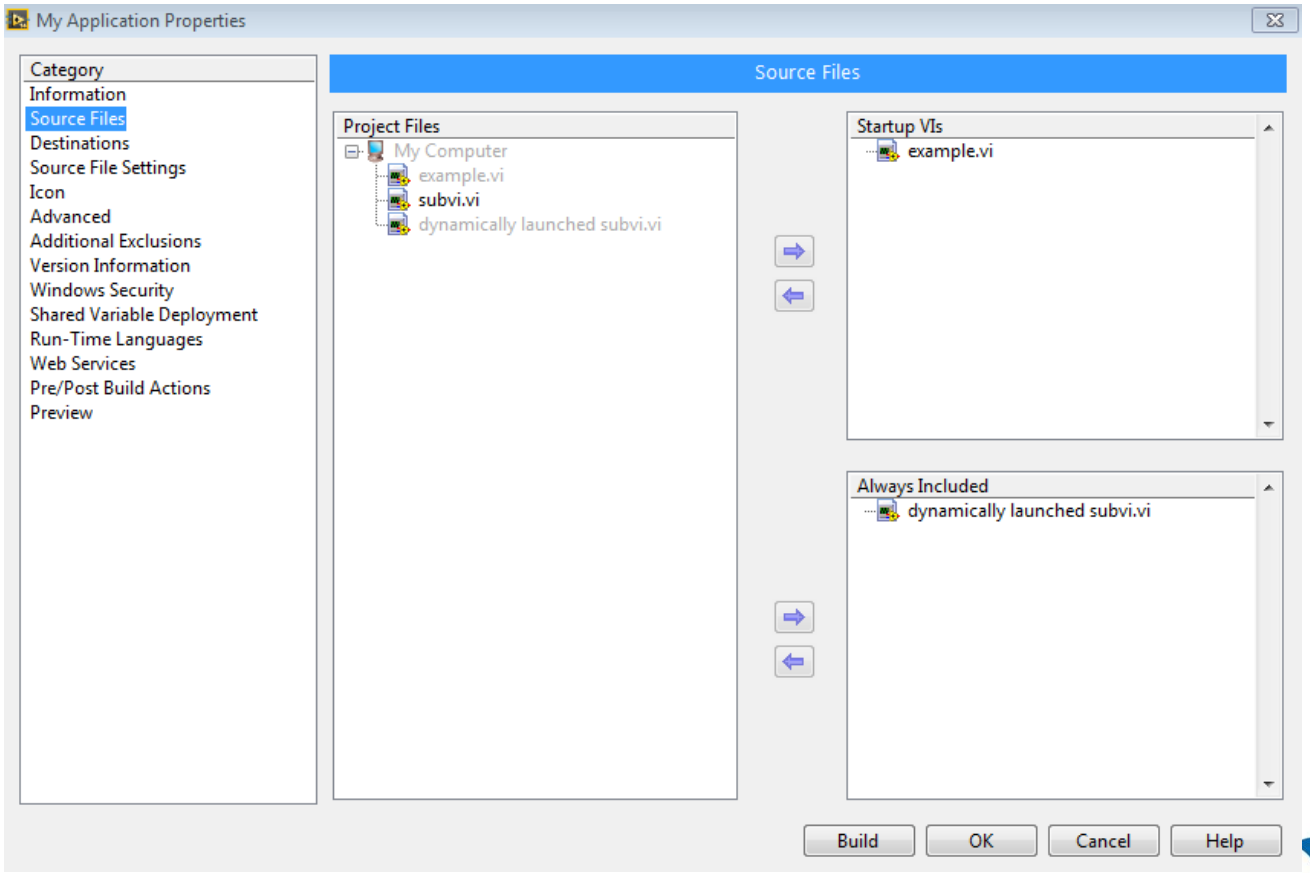


Updating Components – Build Process



Plug-Ins

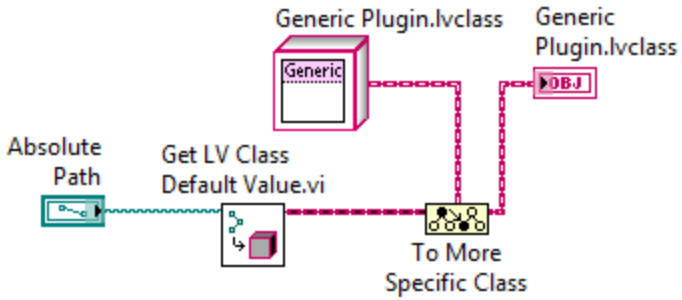
- Extend functionality of existing application without modifying or rebuilding the application itself
- <screenshot of calling VI dynamically>



NATIONAL INSTRUMENTS

Plug-Ins - LVOOP

- Factory Pattern
- Common OO design pattern to instantiate an instance of a class at runtime where the specific class details are not known at edit time
- Must have an abstract class that defines the interface for the plugins
- All plugins must inherit from the abstract class

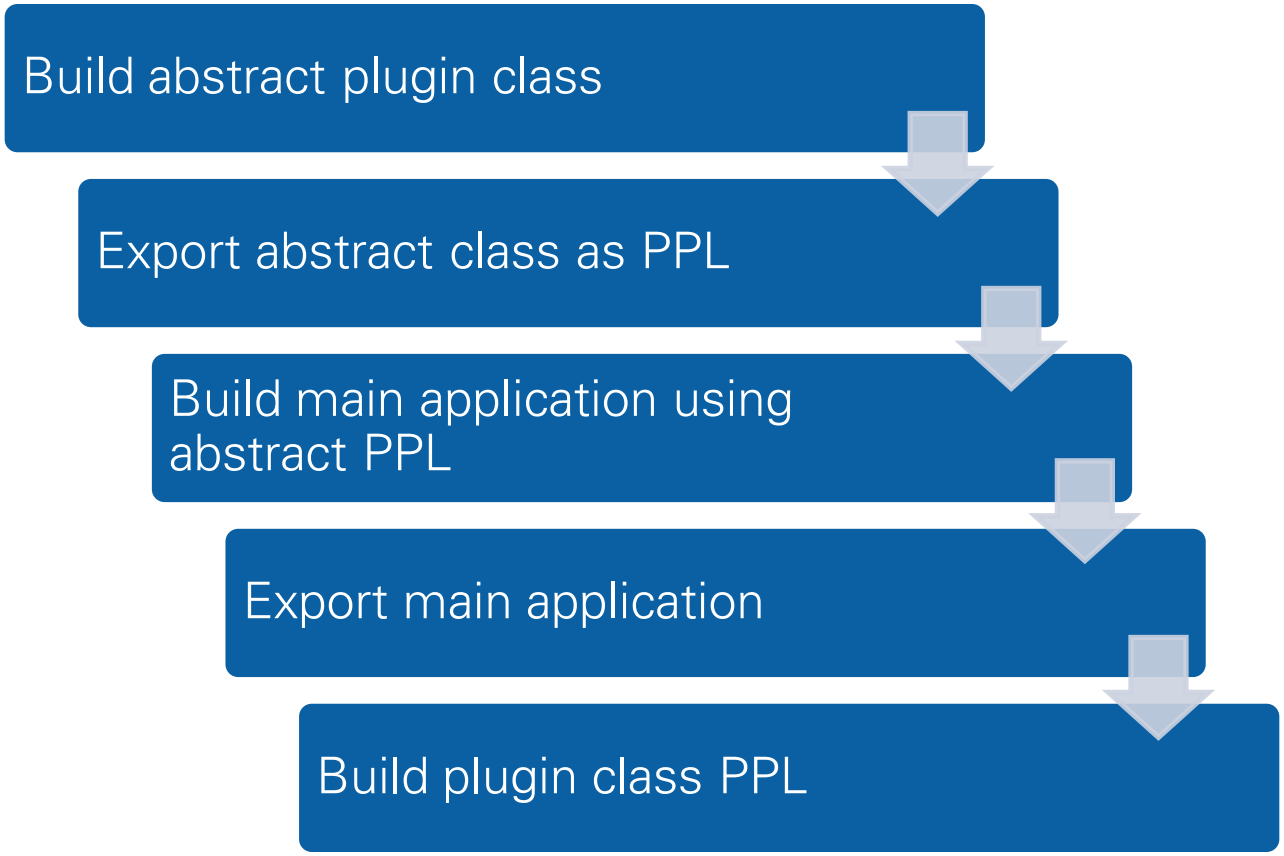


NATIONAL INSTRUMENTS

Packaging Plugins

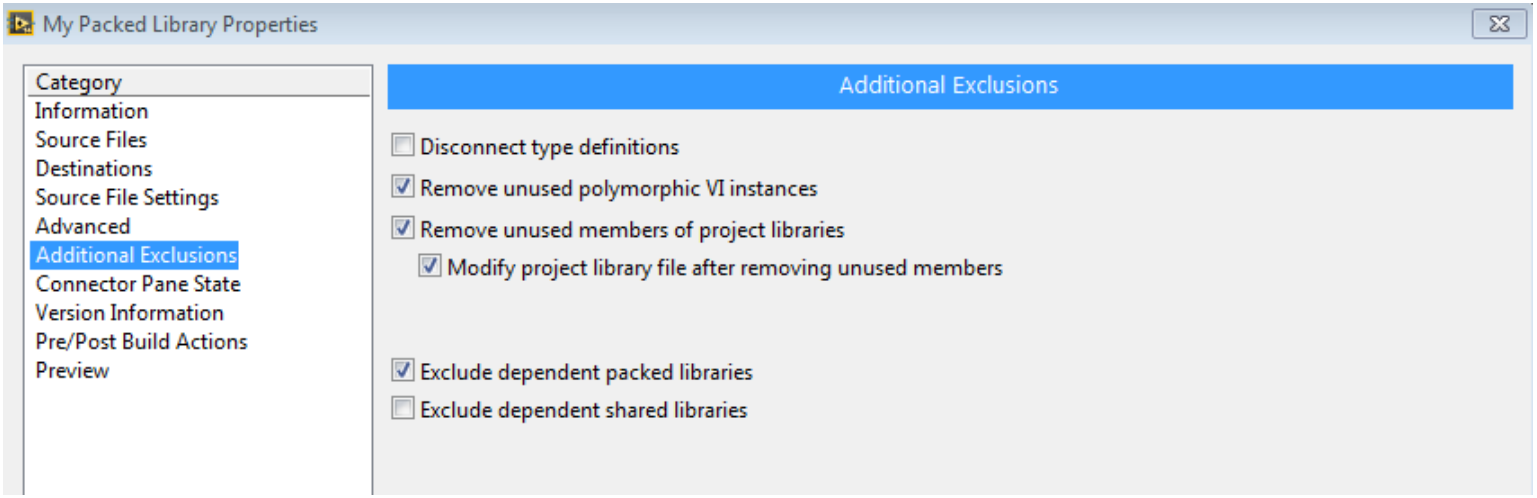
- Two main approaches:
  - Source Distribution/Zip File
    - Simpler build process
    - More files on disk
    - Source code more exposed
  - Packed Project Library (lvlibp)
    - More complex build process
    - Single file per plugin
    - No source code exposed

Packed Library for LVOOP Plugins – Build Process



Packaging Plugins – Dependencies

- Exclude dependent packed libraries from a packed library build (2014 and later)
  - Retains link to original packed library
  - Greatly simplifies creation of plugins (especially LVOOP plugins)



NATIONAL INSTRUMENTS

Considerations

- Consider the target. Source distributions are not platform-specific. Most other types can only be used on one platform (Win, Mac, Linux, LinuxRT, Pharlap, VxWorks, etc)
- PPLs, EXEs, DLLs/Shared Libraries, .NET Assemblies are target-specific