LabVIEW NXG Application Collaboration

2025-03-20

Contents

Collaborating on Web Applications 3
Package Dependencies 3
Sharing a Project and Including Package Dependencies 4
Capturing the Package Dependencies of a Project 4
Resolving Missing or Mismatched Package Dependencies on a Development System 5
Recommendations for Developing a Project Stored in a Source Code Control Repository 6
Source Code Control Models 9
Recommendations for Interacting with a Centralized Source Code Control System while Developing a Project 10
Recommendations for Developing a Project in a Distributed Source Code Control System 11
Merging Files and Resolving Conflicts 13
Code Snippets 14
Creating a Code Snippet 14
Adding a Code Snippet to the Diagram 14

Collaborating on Web Applications

Learn about tools that help your team streamline interactions with project files during software development.

Consider using one of the following tools when working on a project with a team of developers.
These tools are designed to support developer efficiency and preserve the integrity of your code base.

- Package Dependencies Document—Store a list of packages a project uses so you can set up a development system.
- Source Code Control—Back up and manage versions of your source files.

Package Dependencies

In projects, a ***package dependency*** is a package installed on the development system and used in the project.

The ***Package Dependencies*** document (`.sls`) stores a list of packages a project uses so you can set up a development system with the required packages. With the Package Dependencies document, you can do the following:

- Share a project that allows a recipient to easily set up their development system with the package dependencies of the project.
- Create a project that serves as a template. Other developers can use the template project to set up their development system and begin development of the project.
- Update the list of package dependencies any time you add a package dependency to the project. Share the updated list with other developers so they can see new package dependencies you add to the project.

The Package Dependencies document is a Salt State file (`.sls`). For more information about Salt States, visit the SaltStack Documentation website and search for the SALT.STATE.PKG state module.

Related tasks:

- Sharing a Project and Including Package Dependencies

Sharing a Project and Including Package Dependencies

Use the Package Dependencies document to store a list of packages a project uses so you can set up a development system with the required packages.

1. Create a project that uses NI software, drivers, or third-party packages.
2. Capture package dependencies on the source system—Identify the packages a project requires and store a list of those packages in the Package Dependencies document.

   > **Note** When you capture package dependencies, you must have the required packages installed on the source system and the code in the project must be in working condition.

3. Copy the project folder and save it on the target system.
4. Resolve package dependencies on the target system—Set up a development system by installing packages listed in the Package Dependencies document.

Capturing the Package Dependencies of a Project

Identify the packages a project requires and store a list of those packages in the Package Dependencies document.

1. On the Project Files tab, right-click the project and select **Capture package dependencies**.
   The Package Dependencies document opens, scans the project, and displays a list of packages the project requires.
2. (Optional) Make changes to the project, return to the Package Dependencies document, and click **Recapture dependencies**.
3. Click **File** » **Save all**.

After you capture the package dependencies of a project, copy the project folder, save it on another development system, and use the Package Dependencies document to

resolve package dependencies on that system.

Related tasks:

- Sharing a Project and Including Package Dependencies
- Resolving Missing or Mismatched Package Dependencies on a Development System

Resolving Missing or Mismatched Package Dependencies on a Development System

Set up a development system by installing packages listed in the Package Dependencies document.

Before you can resolve package dependencies on a development system, copy the project folder from the system on which the project was originally developed, then save the folder on the target system.

1. Launch the project, then open the Package Dependencies document.
   When you open the Package Dependencies document, it searches the system to identify missing or mismatched packages for the project.
2. Click **Resolve**.
   The Package Dependencies document displays checkboxes next to the missing or mismatched packages. You can review the packages before installing them with NI Package Manager. By default, the Package Dependencies document selects all missing or mismatched packages.

   
   Note NI Package Manager does not support downgrading packages.
3. Click **Resolve selected**.
   The Package Dependencies document launches NI Package Manager.
4. Follow the prompts in NI Package Manager to install the packages you selected.
   Depending on the type of package you install, you may need to restart G Web Development Software or the development system.

Related tasks:

- Sharing a Project and Including Package Dependencies
- Capturing the Package Dependencies of a Project

Recommendations for Developing a Project Stored in a Source Code Control Repository

Because graphical code interacts differently with source code control systems than text-based code does, follow NI's recommendations for developing a project stored in a source code control (SCC) repository.

Note The following information uses examples from Subversion (SVN) and Git. However, you can apply these concepts when interacting with any source code control system.

Recommendation	Details
Create modular code within the project.	***Modularity*** is a software design technique where units of code are collected together into logical and functional groups called ***modules***. Modularity reduces the likelihood that changes to one module of code will introduce unexpected changes to another module of code. To build modularity into your project, create VIs and subVIs that handle specific, limited tasks. Use applications and libraries (`.gcomp`) when you want to group files together to perform a cohesive set of tasks. In team-based development environments, modular code helps create logical divisions of work between developers.
Avoid making simultaneous changes in a VI.	The likelihood of broken code increases when multiple developers modify the same VI

Recommendation	Details
	simultaneously. Merge conflicts are possible when developers submit conflicting changes in a VI. If you use modular design techniques in the project, you can assign modules to individual developers on your team to reduce the likelihood that multiple developers will modify a VI simultaneously. Some source code control systems allow you to lock files when you check them out. Use this feature to prevent others from modifying a VI while you have it checked out. If you cannot avoid simultaneous development of VIs in a project, use NI Compare to view differences between two versions of a VI, then manually combine the differences between each version.
If you make changes to source files in the project, communicate with other developers on your team.	Modifications you make to source files in the project can produce changes to the project file as well as other files. Communicating with your team when you make changes helps other developers to quickly identify conflicts with their work. Using a collaboration tool can help streamline communication on a development team. Some collaboration tools, as well as some source code control systems, can auto-generate emails when changes to the repository occur.
Configure the project directory in the centralized repository to ignore submissions to the following project sub-folders:	Changes within these sub-directories do not affect the functionality of source code in the project.

Recommendation	Details
• `Builds` • `.cache`	In SVN, apply the `svn:ignore` property to the sub-directories. In Git, list the sub-directories in the `.gitignore` file.
Perform code reviews before submitting code to the central repository.	Code reviews improve the quality of the code in your project and preserve the integrity of your source code as it changes during project development. Consider implementing a tiered review framework on your development team. For example: • Peer reviews verify that the submission does not contain defects. • Owner reviews verify that the submission adheres to architectural guidelines. As a code reviewer, you can use NI Compare to compare two versions of a project file or VI.
Create a Package Dependencies (`.sls`) document and keep it up to date during project development.	When you include a Package Dependencies (`.sls`) document in the project, developers can update this file as they create code, then other developers can use it to keep their development systems in sync.
For VIs, disable the auto-merge utility in your source code control system.	The underlying source of a VI file is XML-based text. When merging two versions of a VI, the auto-merge utility might combine lines of text from each version in a way that corrupts the VI and prevents the VI from opening. If using the auto-merge utility on a VI breaks

Recommendation	Details
	code, you may need to revert the VI to an older version to recover your code.
Minimize occurrences of renaming and moving files.	When you rename or move a file, SCC systems delete the file and create a new file with the new name or location you provide. This operation erases any version history of the file with its previous name or location. Other developers could introduce a breaking change when they submit files that use the old name. You might encounter this behavior when you move a VI file to an application or library.

Source Code Control Models

Source code control systems are built using either a centralized or distributed model.

- **Centralized Model**—

  In centralized source code control, a project exists in a single central repository, usually on a server. Users create a local repository on a client where they make modifications to the project, then upload their changes directly to the central repository.

  Apache Subversion® (SVN) is one example of an open-source system using the centralized model.

- **Distributed Model**—

  In distributed source code control, a project exists in a single central repository. Clients have two local repositories: a clone of the central repository and a working repository where users make modifications. Users submit modifications to the working repository often during development. When users finish development work, they push their changes to the central repository.

Git™ is one example of an open-source system using the distributed model.

Recommendations for Interacting with a Centralized Source Code Control System while Developing a Project

Learn how to interact with a centralized source code control provider while developing a project.

Note You can apply these concepts when interacting with any centralized source code control provider.

Checking Out Files and Making Modifications

Recommendation	Details
Lock webVI files to prevent other developers from making simultaneous changes.	Locking webVI files when you check them out reduces the chances of code-breaking merge conflicts.
Update your local copy of the project regularly during development.	Keeping your local copy of the project in sync with the central repository ensures you are developing code against the most recent version of the project. Use the `<centralized source control provider> update` command, where `<centralized source control provider>` is, replace with the name of your provider.
Exit G Web Development Software before updating your remote copy.	If you leave G Web Development Software open when you update your local copy, it is possible to corrupt files in the project.

Checking In Files

Recommendation	Details
Check in all files within the project folder that have changes.	You may notice that some files have changes even if you did not modify them. In particular, the project file will frequently change due to changes in other files. Your source code control provider should scan the project folder for changes. Use this feature to identify all files with modifications.
Ignore the following directories when you submit to the central repository: • `Builds` • `.cache`	Changes within these sub-directories do not affect the functionality of source code in the project. The project directory in the central repository may already be configured to ignore the folders. If not, use commands to configure your source code control provider to ignore the folders.
Add a descriptive comment to your commit.	Descriptive comments help you and other developers interpret changes you submit to the repository.
After you commit files to the repository, notify other team members.	Other developers may need to change their code based on changes you make. Some collaboration tools, as well as some source code control systems, can auto-generate emails when changes to the repository occur.

Recommendations for Developing a Project in a Distributed Source Code Control System

Learn how to interact with a distributed source code control provider while developing

a project.

Note The following information uses terminology and commands from Git. However, you can apply these concepts when interacting with any distributed source code control provider.

Obtaining a Working Copy of a File and Making Modifications

Recommendation	Details
Merge from the central repository regularly during development.	Keeping your local copy of the project in sync with the central repository ensures you are developing code against the most recent version of the project. Fetch and merge to update code in your local repository.
Commit files to your local repository often.	Commit to your local repository as often as you want. It is common to commit locally many times per day.

Committing and Pushing Files

Recommendation	Details
Push all files with changes in the project folder.	Push any files with changes. Most source code control providers scan for changes to the project folder to help you identify changed files.
Ignore the following directories when you submit to the central repository: • `Builds` • `.cache`	Changes within these sub-directories do not affect the functionality of source code in the project. The project directory in the central repository may already be configured to ignore the folders. If not, include the directories in a

Recommendation	Details
	`.gitignore` file. Refer to Git documentation for more information.
Add descriptive comments when you commit to your local repository or push to the central repository.	Descriptive comments help you and other developers interpret changes you submit to the repository.

Merging Files and Resolving Conflicts

Manually editing and merging files helps you maintain stable source code when merge conflicts occur.

Recommendation	Details
Merge project (`.lvproject`) files manually.	Merge project files manually rather than using the auto-merge utility in your source code control system. One expected merge conflict in the project file is the checksum, which records the state of the file when it was last loaded. When manually merging the project file, you can select the checksum from either version. G Web Development Software always reloads the project and generates a new checksum.
Manually edit a VI to resolve conflicts between two versions.	Avoid performing merge operations on VIs. If you identify a conflict between two versions of a VI, view the two versions in NI Compare to locate differences, then edit the code to satisfy the requirements of a project.

Code Snippets

A ***code snippet*** is a PNG image of code that includes the functionality of a VI file.

You can drag a snippet directly onto the diagram and G Web Development Software populates the code depicted in the snippet. You can embed snippets in documents, forum posts, and emails to quickly share and collaborate on code. Snippets are also useful for keeping sections of code that you use frequently handy.

Creating a Code Snippet

Use code snippets to store and share G Web Development Software code as PNG files that become graphical code when you drag them onto the diagram.

1. Open a project.
2. On the diagram, highlight the code you want to capture as a snippet.
3. Click **Edit** » **Create snippet from selection**.
4. Choose where to save the snippet, give it a descriptive name, and click **Save**.

> **Note** If you edit a snippet in an image editor, the editor will remove the VI information from the file and you will not be able to drop the snippet onto the diagram.

> **Tip** Sharing snippets is most useful for simple code. If your snippet includes dependencies, such as subVIs or classes, anyone who uses your snippet must have those dependencies in their project or they will see a missing dependency error.

Adding a Code Snippet to the Diagram

Use code snippets to drag sections of code onto your diagram.

Before you begin, make sure you have any subVIs included in the code snippet in your project.

1. Save the snippet to your machine.

> **Note** If you edit a snippet in an image editor, the editor will remove the VI information from the file and you will not be able to drop the snippet onto the diagram.

2. Open your project and navigate to the diagram of the VI you want to add the snippet to.
3. Drag the snippet from its location on your machine to the diagram.
   The code depicted in the snippet appears on the diagram.