Sharing Code with the LabVIEW Project Library

Publish Date: jul 06, 2018 | 25 Ratings | **3.28** out of 5 | Print | 1 Customer Review | Submit your review

Overview

This article is part of a series on software engineering practices and tools for large application development in LabVIEW.

Click here to view the list of other articles

As an application grows larger, it might become challenging to share and maintain code among teams of developers. Sharing project code is facilitated by defining a well-specified public interface for each component in a large application and then hiding the implementation details of each component from the other parts of the system. If a certain component has a public interface that other components communicate with, a developer can change the underlying implementation details at any time without affecting the other components that depend on it. This process saves development time and keeps applications easy to maintain and debug. Moreover, this process leads to standardized, well-organized, and well-managed code.

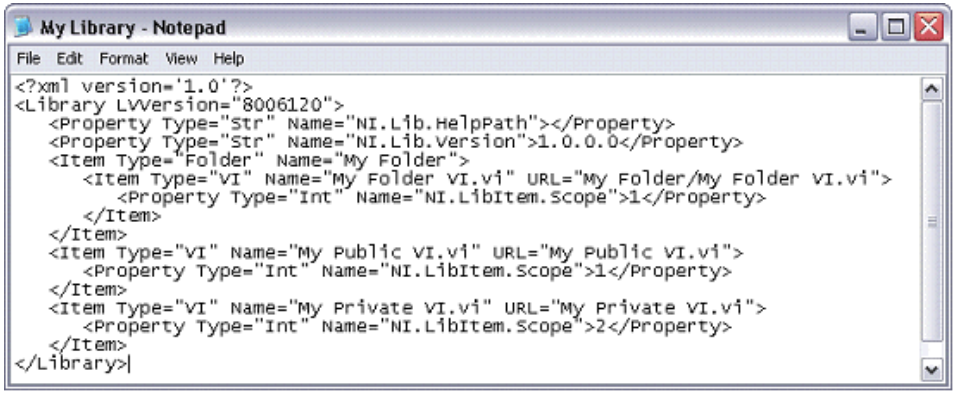
The type of programming described above can be achieved in LabVIEW through modular programming. Modular programming is the breaking down of an application into many individual tasks that you can independently run, manage, and reuse. LabVIEW is an ideal environment for building modular code. Users are encouraged to build subVIs when there is a logical division of labor in their application. LabVIEW users sharing code should store all shared code in a common repository and track changes to the code using source control tools. Also, the LabVIEW project library, introduced in LabVIEW 8.0, enables users to place restrictions on different VIs in the application to aid them in designing, developing, and maintaining large projects.Refer to the LabVIEW Development Guidelines book in the LabVIEW Help for more information about source control, modular programming, and other development guidelines. This document provides an overview of the project library and discusses its role in sharing and reusing code.

Table of Contents

1. What is a Project Library?

LabVIEW project libraries are collections of VIs, type definitions, shared variables, palette files, and other files, including other project libraries. If you create and save a new library, LabVIEW creates a project library file (.lvlib), which includes the properties of the project

library and the references to files that the project library owns.



[+] Enlarge Image
Figure 1. Project Library XML File

A project library is represented as a sub-tree within the Project Explorer window. You can create a project library in a LabVIEW project by right-clicking on the **My Computer** icon and selecting **New»Library** from the shortcut menu. Refer to Using Project Libraries in the LabVIEW Help for more information about creating and using project libraries.

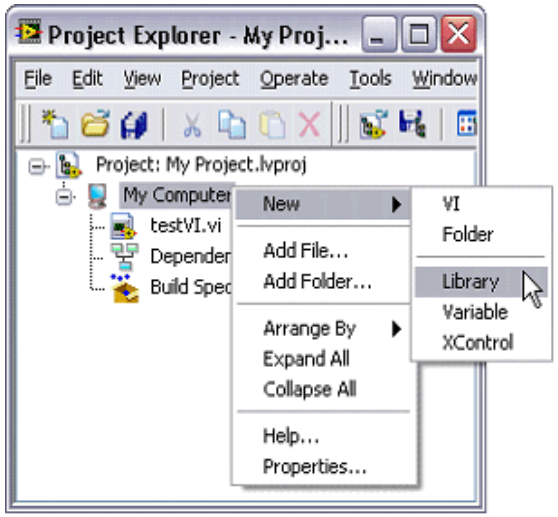


Figure 2. LabVIEW Project Containing a Project Library

2. Benefits of a Project Library

Back to Top

Project libraries provide two main benefits:

Namespace

Project libraries provide a namespace qualifier for VIs and other files that belong to the library, which means that the name of the project library is prefixed to the name of all member VIs and files. For example, consider a large application that includes two different instrument drivers, InstrDriver1 and InstrDriver2, who's VIs are respectively organized in project libraries calledInstrDriver1.lvlib and InstrDriver2.lvlib. Assume that each of the instrument drivers contains a unique VI called Initialize.vi.

The names that LabVIEW uses for these VIs would be InstrDriver1.lvlib:Initialize.vi and InstrDriver2.lvlib:Initialize.vi. You can call both of these VIs as subVIs from the same top-level VI without any problems.

The namespace helps make the VI names more unique to alleviate naming collisions, which is important as applications grow larger and contain many VIs. In fact, the underlying structure of instrument drivers of National Instruments have been organized in project libraries since LabVIEW 8.

Controlled Interface
Within a project library, individual members or folders can be designated at edit-time as either **public** or **private**. Public VIs can be called from any other VI and are any VIs you want users to call directly. Private VIs can only be called as subVIs by other VIs in the same library and would include support VIs you might want to edit later without taking the risk of breaking users' code.

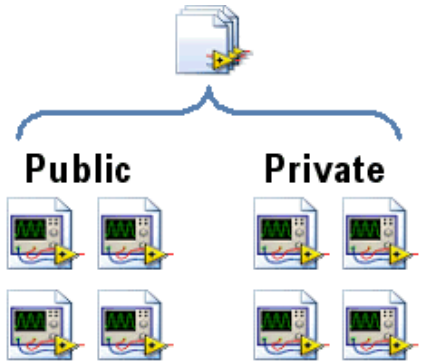


Figure 3. Structure of a Project Library

In a typical code module packaged as a project library, public VIs include the top level VIs that provide interface to the module, whereas private VIs are low-level support VIs that hide the implementation details of the code module. Project libraries restrict/protect users from explicitly calling private VIs. Users are forced to use public VIs in their applications and the application breaks if they attempt to use private VIs. After distributing the project library, developers can make changes to the private VIs, but as long as they do not change the public VIs, they can safely distribute the new versions of the code module to their users. Users' existing code that makes use of the code module do not break and work as expected with the new version. Thus, with project libraries applications become easier to maintain. Furthermore, collaborative development is safer because developers can restrict access to VIs that are not meant to be used by other developers, while allowing the usage of intended VIs by making them public.

Project libraries can be viewed and edited through the Project Explorer. If you add a VI that is contained in a library to a project, the entire library is be added to the project tree. Also note that any VI within a library never appears in a project tree except under its library. As mentioned earlier, instruments drivers are packaged in LabVIEW 8 as project libraries. Figure 4 shows a project library named Tektronix TDS 1000 2000 Series.lvlib from an instrument driver. Note that the VIs in the library are organized as public or private.

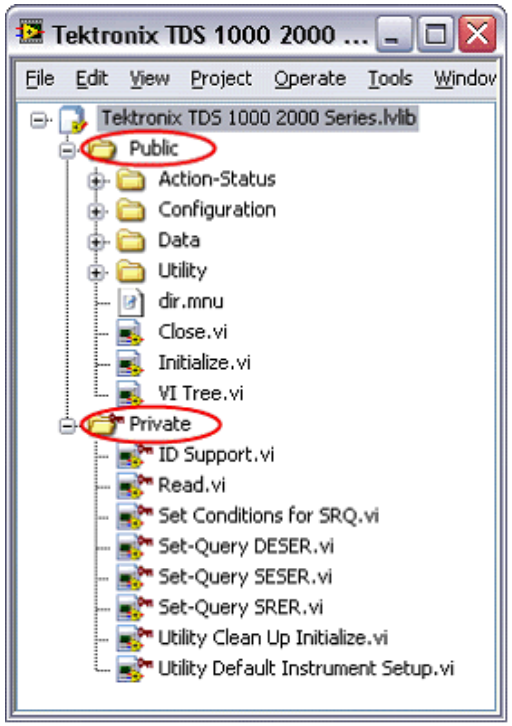
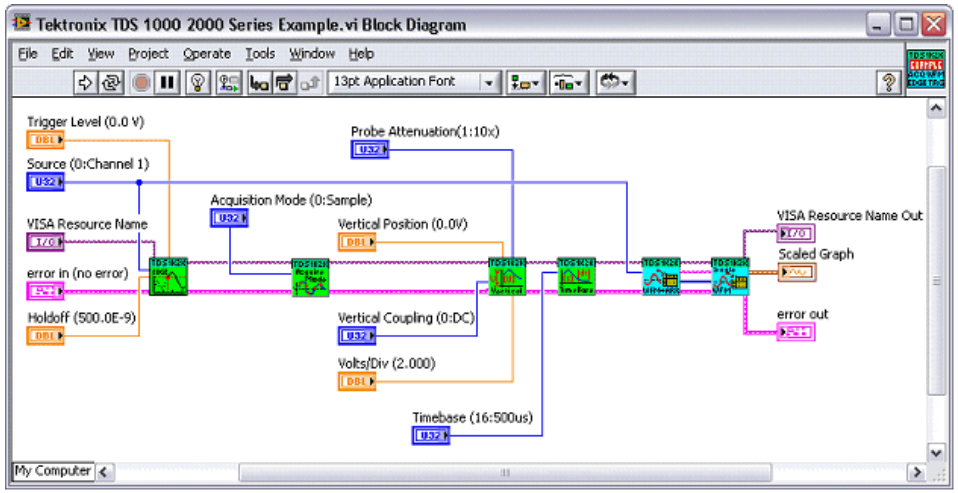


Figure 4. Project Library for a Tektronix Oscilliscope

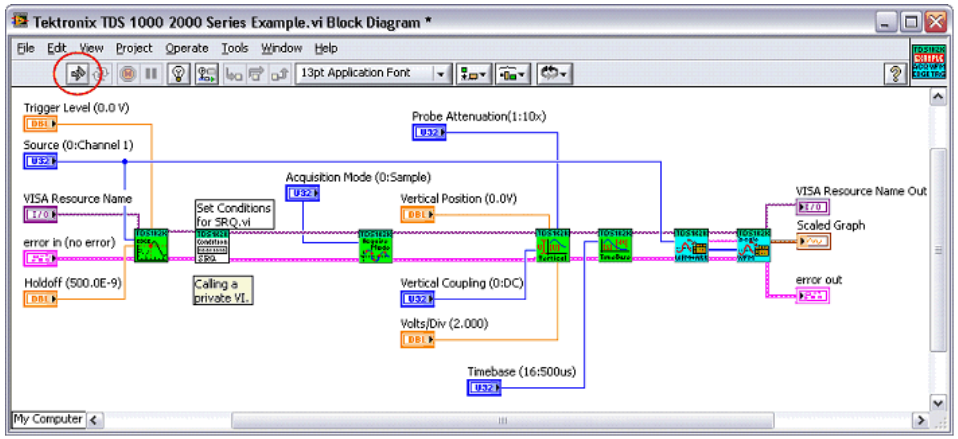
Users can explicitly call the public VIs from this library as shown in the Tektronix TDS 1000 2000 Series Example.vi in Figure 5.



[+] Enlarge Image

Figure 5. Example VI that Uses Public VIs from Tektronix TDS 1000 2000 Series.lvlib

However, as soon as Set Conditions to SRQ.vi is added to the block diagram, the run arrow breaks (Figure 6), because Set Conditions to SRQ.vi is defined as a private VI in the project library (Figure 7):



[+] Enlarge Image

Figure 6. Example VI Calling the Private VI Set Conditions to SRQ.vi

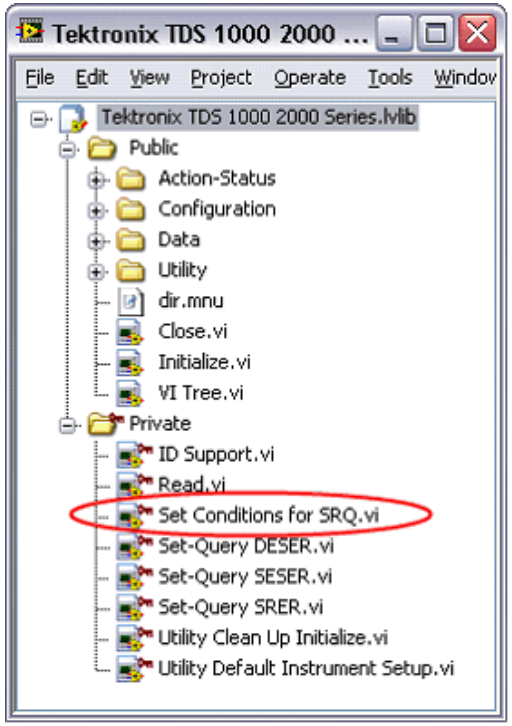
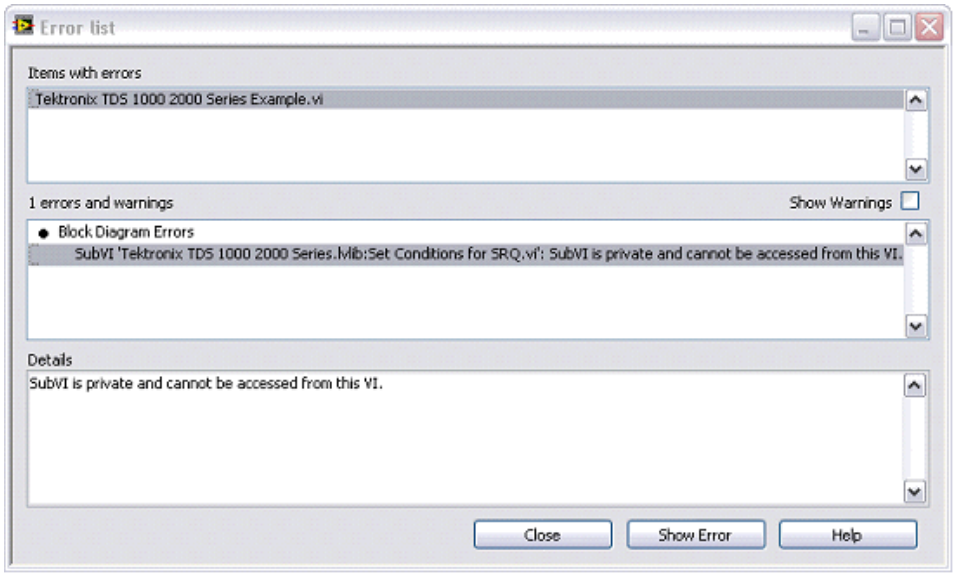


Figure 7. Set Conditions to SRQ.vi in a Project Library

The resulting error message states that **SubVI is private and cannot be accessed from this VI**(Figure 8). This shows how LabVIEW enforces that private VIs cannot be called from VIs outside the library.



[+] Enlarge Image

Figure 8. Error Explanation

3. Configuring a Project Library

Back to Top

You can configure the editing permissions and access settings for project libraries by right-clicking on the library and selecting **Properties**.

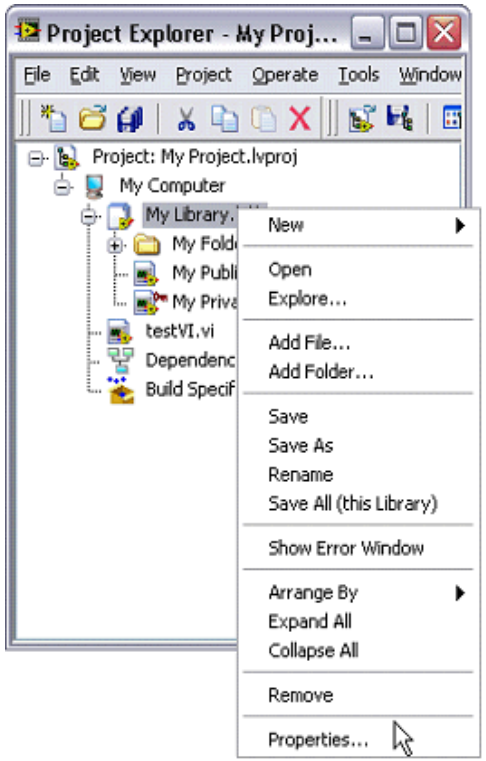
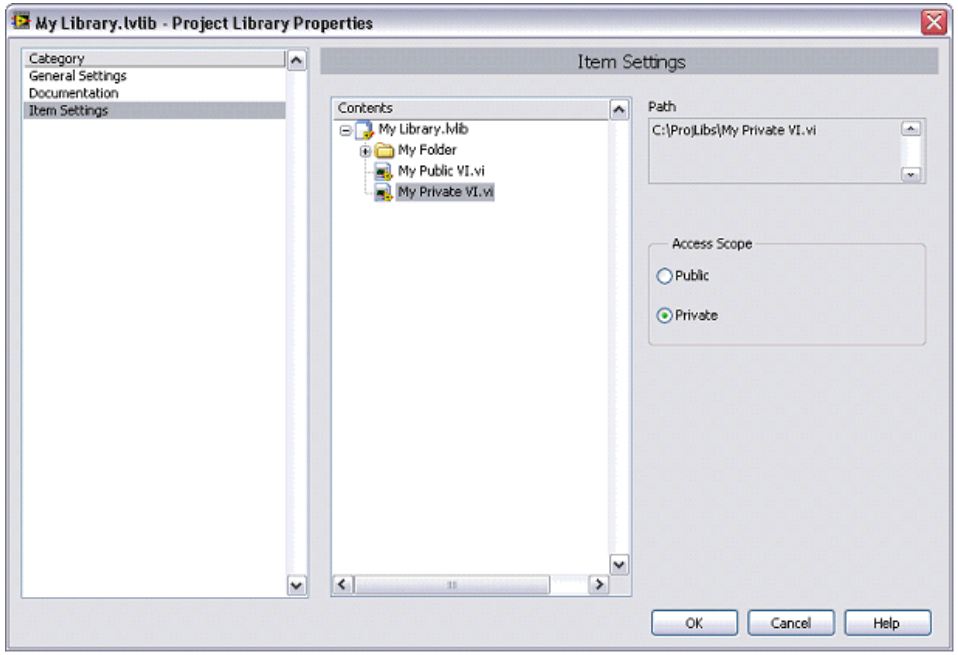


Figure 9. Accessing Project Library Properties

The **Project Library Properties** configuration window provides a central configuration point for an entire project library. Through this window you can elect project library members as public or private and apply various protection levels to the project library. These properties are separated from the actual code (items in the library), so unlike in text-based programming languages, you do not have to modify the contents of each program to change each program's access scope.

Under the **Item Settings** in the **Project Library Properties** configuration window, you can set the access scope of the items and folders in the project library.



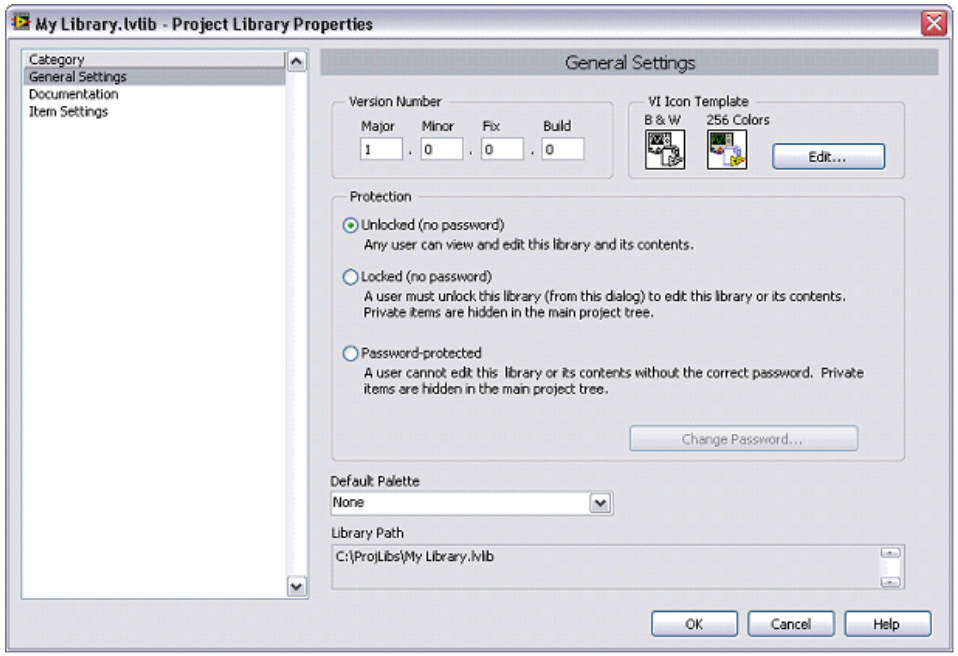
[+] Enlarge Image

Figure 10. Project Library Item Settings Dialog

If a VI is marked as **public**, other VIs and applications can call it. If a VI is designated as **private**, other VIs and applications outside the project library cannot call it. If you have a folder within a project library, you can set the access scope to public or private, which affects the entire contents of the folder. In addition, you can select the **Not specified** option for folders, in which case you can set access scopes for individual
items within a folder on a case-by-case basis.

Setting the access scope for the project library offers users a protection from incorrectly using code. Also, if developers make changes to their code thereafter, they know which parts of the code affects the users, depending on whether they changed the private or public VIs in the project library.

Under the **General Settings** in the **Project Library Properties** configuration window, you can set the version number, select an icon, and set levels of protection for the library.



[+] Enlarge Image

Figure 11. Project Library General Settings Dialog

You can apply varying levels of security for the project library by setting the protection level to **Unlocked**, **Locked** or **Password-protected**. If the project library is locked, users cannot add or remove items, and cannot view the items in the library that are set to private. This option is useful if you are developing a project library and do not want the end user to be tempted to try to use its private files. If the project library is password protected, users do have to supply a correct password to edit the library and to view the private items it owns. However, the public VIs in any project library can be called regardless of the protection settings. It is useful to password protect a project library if you want only select people on a development team to have editing privileges to the library. However, note that adding password protection to a project library does not add password protection to the VIs it owns (see the FAQ below for more information).

The **Documentation** category under the **Project Library Properties** window lets you supply a localized name, description, help path and help tag for the library. The description you enter for the library shows up in the Context Help if you hover over the library in Project Explorer.

4. Recommended Practices for Using a Project Library

Back to Top

It is recommended that public items in project libraries are determined during the**design phase** of a project. For example, if there are multiple developers working on the same project, they should agree on the interfaces they will use to communicate between different code modules in a system. Components that developers wish to share with others (i.e., public interface to a set of VIs) can then be designated as public VIs in project libraries. Developers can create private VIs as needed which cannot be called outside of the library, but provide the actual implementation details of the code module.

5. Frequently Asked Questions

Back to Top

What is a difference between a project library and an LLB?
An LLB is a container for a group of VIs saved on a single binary file. A project library is an XML file containing references to the files within it. Unlike an LLB, a project library does not actually represent a space on disk where all the files are saved.

LLBs do not provide an organization or hierarchy to the VIs within them, aside from letting users declare one or more VIs as top-level. Project libraries, in contrast, offer a hierarchy for the VIs within them through the use of folders and sub-libraries. Also, project libraries apply a namespace to the items they own to overcome naming collisions for VIs, whereas LLBs do not. Finally, project library members can be either public or private, restricting how they are used, while access to members of an LLB is completely unrestricted.

Can a project library contain an LLB or vice versa?
Project libraries can contain LLBs like any other files, and LLBs can contain project libraries. However, if you add a project library to an LLB, the project library file (.lvproj), not the items project it owns, is added to the LLB.

What is a difference between a project library and a folder in a project?
Project libraries are similar to folders in a project in that they are ways of grouping VIs together for organization. However, whereas libraries create a namespace qualifier for a group of VIs, folders are simply a visual organization within the Project Explorer window and do not designate a namespace to the VIs within them. This means that folders do not alleviate naming collisions the way that project libraries do. Also, a folder and the VIs within it cannot be defined private like the VIs within a library, unless the folder is within a library.

Can multiple project libraries own the same VI?
The same way that a project library contains a reference to the VI within it, the VI also contains a reference to its owning library. A VI can contain a reference to only one library, so only one project library can own a particular VI.

How do I distribute a project library?
In order to distribute a project library, you can simply distribute the .lvlib file along with all the other files the library owns. Alternatively, you can create a single zip file containing the entire library and distribute it. To accomplish that right-click **Build Specifications**, select **New»Zip File**, and include the project library under the **Source Files** category. For more information on creating zip files refer to Distributing Applications with the LabVIEW Application Builder.

Does password protecting a project library add password protection to the VIs it owns?
Adding a password protection to a project library does not automatically add password protection to the VIs it owns. As a developer, you might want to password protect a project library if you do not want the users to view the library's private items in the **Project Explorer** window or to edit the library. But this does not mean that users do not have access to the private VIs of the project library. Project libraries are intended to package code to provide controlled interface to their members. However, libraries do not restrict users from seeing how the code works. Users can still open the private VIs and modify them, and as long as these private VIs are executable and users don't change their connector panes, other VIs that call them do not break. However, if you do not want users to be able to view or edit private VIs altogether, you need to password protect them separately.

Where is the password information stored for project libraries?
If the project library is password protected, the password information in stored in the library file in an encoded form.

How does a project library link to the VIs it owns? If I later change the physical location of the VI in my project library, can my library find it?
A project library is an XML file and for each item the library owns there are attribute names called Name and URL. The attribute value for Name contains the name of the item (e.g., My VI.vi), and the attribute value for URL contains a path to the item relevant to the location/path of the library itself.

If a VI within a project library is moved to a different location on disk and you try to open it, LabVIEW begins searching for it. Once the VI in a new location is found (automatically or by you browse to it), LabVIEW remembers the new location and updates the project library.

How do the member VIs link to their project library?
A VI contains a reference to its owning project library and this information is encoded as part of the binary file of the VI. If you wish to detach a VI from its project library from within a VI itself, you can open that VI and select **File»Disconnect VI From Library** option. This option can be very useful if you have access to the VIs, but loose the project library file.

Since a project library is an XML file, can I edit the library by editing the XML file properly?
You can corrupt the library, at least if you add a new VI to the library by appropriately editing the file, since the VI has to have a link to the library as well, and users cannot do that.

If I add a VI to project library, will all of its subVIs also be added to the library?
If you add a VI to a project library, its subVIs are not automatically added to the library.

Can libraries contain other libraries? If so, how do the access settings of the project library apply to member libraries?

Project libraries that another project library owns are called project sublibraries. Project sublibraries are useful if you want to create a project library that includes separate areas of functionality. For example, if you are creating a project library of graphics tools, you might divide the two-dimensional and three-dimensional drawing tools into separate sublibraries. Each project sublibrary can include private and public access items and folders.
The settings in the owning project library do not affect not access settings and editing permission for items within the project sublibrary. You can set the access of a project sublibrary file (.lvlib) as private within the owning project library, but if you edit the project sublibrary itself, items the sublibrary owns retain public or private access settings.

Can I create or edit project libraries programmatically?
Through the VI Server interface, you can get or set different properties for project libraries and invoke different methods on them. Figure 12 illustrates a subset of methods available for project libraries under the Library class as shown in the Class Browser.

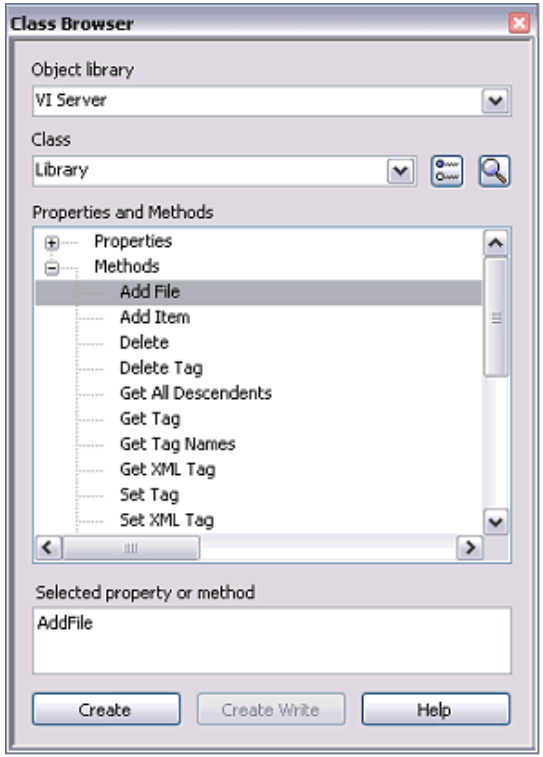


Figure 12. VI Server Interface for Project Libraries

Related Links:

LabVIEW Help: Development Guidelines
LabVIEW Help: Using Project Libraries

Back to Top

Customer Reviews

Customer Reviews

1 Review | Submit your review

Improvements to page - 18-jul-2007

It is very difficult to write good software documentation. If the documentation is comprehensive, it is too difficult and complex to understand when you don't know anything about what it is trying to explain. If the documentation introduces concepts in simple steps, it becomes too long for the experienced user to digests and find information he or she is looking for. Like lots of mediocre software documentation, it is only possible to understand this page once you know everything about what it is describing. I found it quite a struggle. In the end it did not describe what I was looking for and I had to find this by trial and error. We make use of the old LabVIEW 7.1 'save with options' facility to create a LabVIEW library consisting of a single top level module but including all the support the eyes required to run it, including fro vi.lib. This gets dynamically loaded into a parent application to provide control panels specifically tailored for individual remote laboratory tasks (see http://telelrobot.mech.uwa.edu.au/) It was very difficult to figure out how to do this in LabVIEW 8.2. Eventually I figured out that you have to create both an 'lvlib' and build a 'distribition' to create the 'llb' that is needed. I don't think that you describe this step in the documentation that you provide. I think it is great that you have moved to automate the production of complex LabVIEW applications: it