Document Type: Tutorial
NI Supported: Yes
Publish Date: Jul 28, 2011

Developing Portable Shared Variable Applications

Overview

This tutorial walks the developer through creating portable applications that communicate with Shared Variables.

Table of Contents

Introduction

When developing an application with a Shared Variable Library, there are additional considerations to take into account to build an executable and deploy to other systems.

Background

A shared variable is accessible through a network, but hosted on a single machine. When developing your VI in LabVIEW, the shared variable library is automatically deployed to the target it is listed under in the project explorer. When the library is deployed to this target, that target is now "hosting" the shared variables contained in the library. Other computers will have to connect to this target to read or write the shared variable library.

In Figure 1, because the library is located under My Computer in the project explorer, My Computer will host the shared variables. You click and drag this library to sit under a different target to change the intended library host.

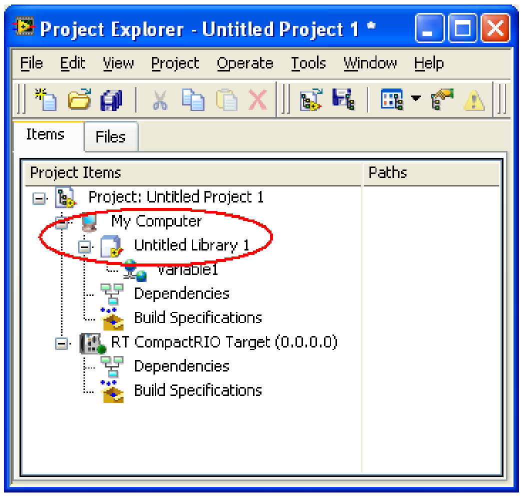

Project Explorer - Untitled Project 1 *
File Edit View Project Operate Tools Window Help
Items
Files
Project Items
Paths
Project: Untitled Project 1
My Computer
Untitled Library 1
Variable1
Dependencies
Build Specifications
RT CompactRIO Target (0.0.0.0)
Dependencies
Build Specifications

Figure 1

It is important to note a shared variable library is not actually hosted by the target until the library is deployed. If you try to read from a variable that has not been deployed, an error will result. You can manually deploy a library in the LabVIEW development environment by right clicking the library and selecting deploy. When a VI with a shared variable on the block diagram is run (with the run button), the library is 'auto deployed' by LabVIEW. Auto deploy is enabled by default at the library level and can be disabled if desired in LabVIEW versions 8.0 to 8.6.1. The option is seen in Figure 2.

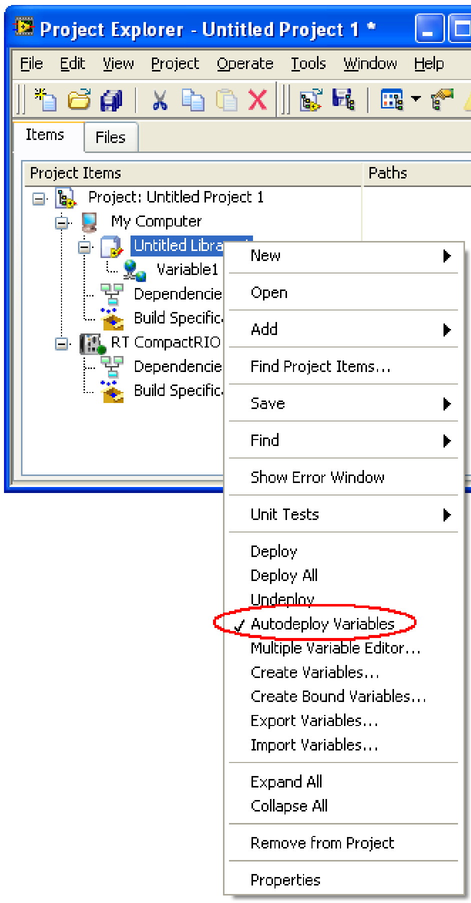


Figure 2

In LabVIEW 2009, the auto deploy option was moved to the target level and can be seen in Figure 3.

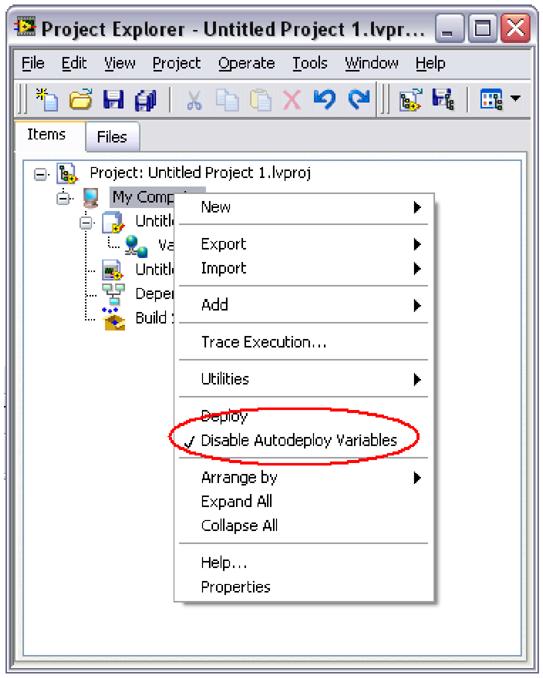

Project Explorer - Untitled Project 1.lvpr...
File Edit View Project Operate Tools Window Help
Items Files
Project: Untitled Project 1.lvproj
My Comp
New
Export
Import
Add
Trace Execution...
Utilities
Deploy
✓ Disable Autodeploy Variables
Arrange by
Expand All
Collapse All
Help...
Properties

Figure 3

Only one target hosts the library. This library can be accessed by other targets on the network by simply creating a VI in the same project, and dragging and dropping the variables on to the block diagram. If the device connecting to the library is a host computer, create this VI under My Computer. If the device connecting to the library is a LabVIEW Real-Time (RT) target, create this VI under the RT target.

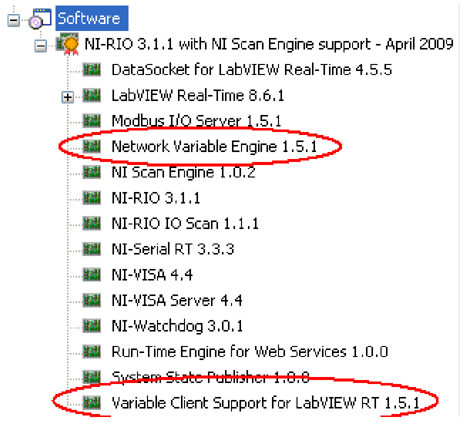
Where to host a Shared Variable Library?

When developing an application with shared variables there are inevitably multiple devices connecting to the library including PCs and NI PACs. Typically all of these devices are also capable of hosting the library. The optimal location to host shared variables depends upon the application

Hosting shared variables takes up memory and processor time on the host.

If your application is limited to multiple computers running Windows there is little difference to where the library is hosted. You may want to decide this for yourself based upon the availability and relative performance of the host computers.

If your application includes a single host computer and a single or multiple RT targets it is recommended to host the library on the host computer. This will save memory and processor time on the RT targets. Also, by hosting the library on the host computer instead of the RT target, you can reduce the software install footprint on the RT target. The RT target will not need the Network Variable Engine, but will still require Variable Client Support. The Network Variable Engine allows hosting, the Client allows communicating with libraries on other machines. You can see a RIO target with these installed in Figure 4.


Software
NI-RIO 3.1.1 with NI Scan Engine support - April 2009
DataSocket for LabVIEW Real-Time 4.5.5
LabVIEW Real-Time 8.6.1
Modbus I/O Server 1.5.1
Network Variable Engine 1.5.1
NI Scan Engine 1.0.2
NI-RIO 3.1.1
NI-RIO IO Scan 1.1.1
NI-Serial RT 3.3.3
NI-VISA 4.4
NI-VISA Server 4.4
NI-Watchdog 3.0.1
Run-Time Engine for Web Services 1.0.0
System State Publisher 1.0.0
Variable Client Support for LabVIEW RT 1.5.1
Figure 4

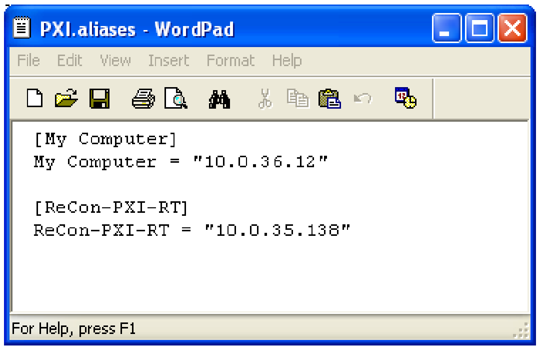
If your application includes multiple host computers and a single RT target, it is recommended to host the variables on the RT target. This allows for easier deployment and scaling to additional host computers, because the data is always on the RT target. Make sure the Network Variable Engine is installed on the RT target as shown in Figure 4.

How will my deployed executables know what computer or RT target is hosting the library?

When running VIs in the LabVIEW development environment, the location of the library in the project dictates where each VI looks for the hosted shared variable. After building your VIs into executables, and moving your exe's to a different host computer and/or RT target, your application may not continue working as before.

For instance, if your RT target is communicating with shared variables on a host computer, and you change host computers, your RT target would need to be informed of this change. The same goes for moving any application from a computer or RT target to another computer or RT target. The applications need to know where to get their data.

When a project is saved and when an executable is built, an additional file is created with a .aliases extension. This is a text file (editable in notepad) that defines the network addresses for each target. A sample .aliases file can be seen in Figure 5.



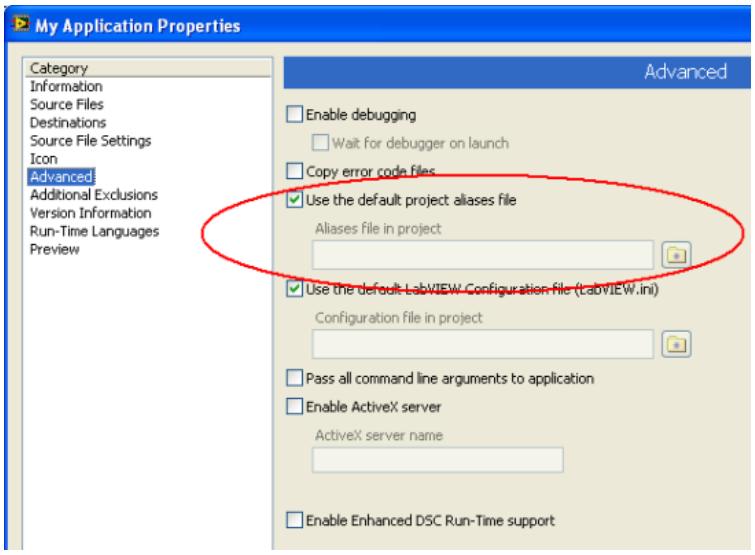
```
[My Computer]
My Computer = "10.0.36.12"

[ReCon-PXI-RT]
ReCon-PXI-RT = "10.0.35.138"
```

Figure 5

By modifying the .aliases file that accompanies your .exe, .rtexe, or .lvproj, you can direct that application to look at the appropriate network address hosting your library.

For example, the .aliases file in Figure 5 contains two targets. A target named "ReCon-PXI-RT" and a target named "My Computer." If this project built two executables, one to go on the host computer and one to go on the RT target, there would also be a .aliases file accompanying the generated .exe and .rtexe. When the .exe attempts to read from the library it will consult the .aliases file that accompanies it and try to contact 10.0.36.12. The same goes for the .rtexe on the RT target. If the developer decided to change host computers, the network address in the .aliases file accompanying the .exe and .rtexe would need to be changed to reflect the new address of the new host computer.

LabVIEW application builder includes tools to make this process easier. When building an application, the developer can define if that application will use the default .aliases file with the project or use a different one. If the host application is intended for a different PC, the developer can specify a custom .aliases file for that application to build with. This can been seen in Figure 6.


My Application Properties

Category: Information, Source Files, Destinations, Source File Settings, Icon, **Advanced**, Additional Exclusions, Version Information, Run-Time Languages, Preview

Advanced

- [ ] Enable debugging
  - [ ] Wait for debugger on launch
- [ ] Copy error code files
- [✓] Use the default project aliases file
  - Aliases file in project
- [✓] Use the default LabVIEW Configuration file (LabVIEW.ini)
  - Configuration file in project
- [ ] Pass all command line arguments to application
- [ ] Enable ActiveX server
  - ActiveX server name
- [ ] Enable Enhanced DSC Run-Time support

Figure 6

After modifying the .aliases file, the executables will connect to the network address listed to find the shared variable library. This target must have the library deployed to it or else an error will be reported.

How will the library be deployed to the target machine?

As seen above, shared variable libraries are auto deployed (by default) from the project when in the LabVIEW development environment. This is **not the case** in a compiled executable. The developer will have to take this into account to ensure the shared variables are on the desired target.

Shared variables can be deployed programmatically from within your compiled executable with the application level method seen in Figure 7. Note that this method is unavailable on RT targets, however, this method can be used to deploy to a RT target if an IP address is specified.

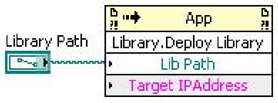
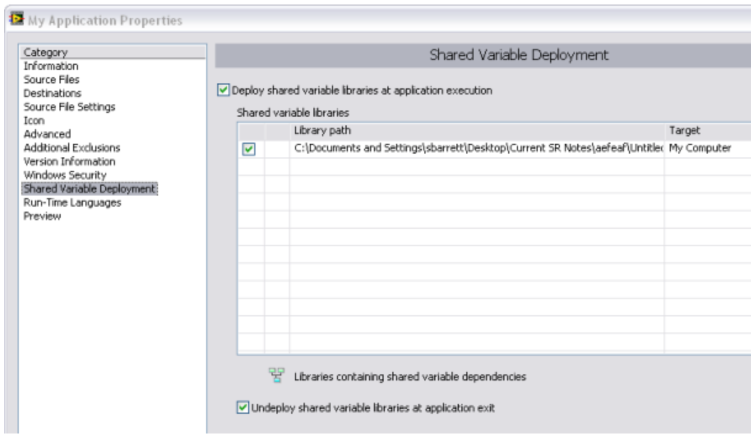


Figure 7

In LabVIEW 2009, a new option was added to the application builder to ease this process. The LabVIEW 2009 application builder can build your existing code into an application, and handle programmatically deploying and un-deploying your shared variables for you. After adding your library to the always included source file list as seen in Figure 9, you can configure this library's deployment options as seen in Figure 8.


My Application Properties

Category: Information, Source Files, Destinations, Source File Settings, Icon, Advanced, Additional Exclusions, Version Information, Windows Security, **Shared Variable Deployment**, Run-Time Languages, Preview

Shared Variable Deployment

- [✓] Deploy shared variable libraries at application execution

Shared variable libraries

	Library path	Target
✓	C:\Documents and Settings\sbarrett\Desktop\Current SR Notes\aefeaf\Untitlec	My Computer

Libraries containing shared variable dependencies

- [✓] Undeploy shared variable libraries at application exit

Figure 8

Additionally, when building a RT application with the RT application builder, you can specify that the application includes a shared variable library. Upon building and deploying this application, the shared variable library will be deployed to the RT target. This can be seen in Figure 9. This allows a developer to deploy an application and a shared variable library to a RT target at the same time.

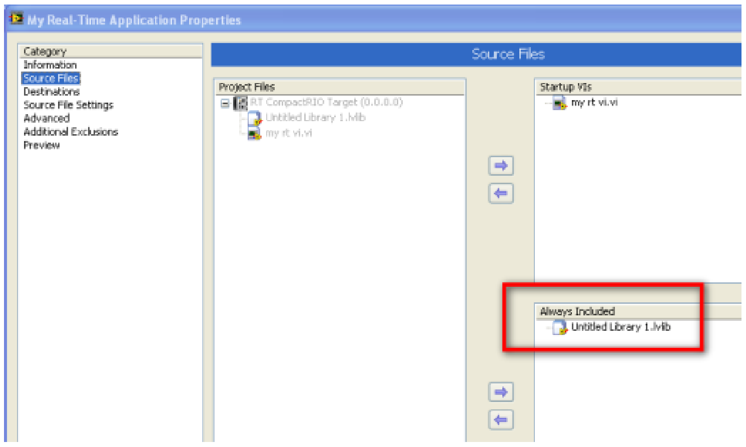


Figure 9

Summary

In conclusion, a developer can ensure that the executable is looking at the correct device for shared variable communication by modifying the .aliases file. The developer can also ensure that device is hosting the library by programmatically deploying from their application or deploying with the application builder.