

Development Distribution Tool

White Paper

Mattias Ericsson

<sy**mbio**>

Abstract

The *Development Distribution Tool* is used for distributing LabVIEW™ code. VIs, type def controls and menus are copied into the distribution that might either be a LLB or a folder.

What is the difference between the standard LabVIEW™ “Save with options...” and this new tool?

- All VIs, type def controls and menus might be *prefixed*.
- Distribution might be stored in a *LLB or folder*
- There might be *more than one top-level VI*.
- Top-level VIs might be saved *outside the LLB*.
- Exclude paths. *Any VIs might be excluded* from the distribution.
- *Dynamically called VIs by a VI server and their sub-VIs* might be included.
- GOOP 2.0 virtually called sub class methods are *automatically added*.

The *Development Distribution Tool* with all of these new features *makes it really easy to distribute LabVIEW™ code* in a professional way.

Prefixing minimizes the risk of accidental cross-linking between source code and distribution! With prefixing is also possible to *run distribution of different versions at the same time*.

The included support tool, *Create Build Script*, creates a build script (.bld file) to use with the LabVIEW™ Build Application Tool (for building .exe files). If the application contains any GOOP 2.0 class that virtually calls any sub class method, the *Create Build Script* tool analyzes the application and automatically adds all needed dynamic VIs in the build script.

System requirements are LabVIEW™ 6.0.2 (or later) and runs on all platforms. To use the build script file created by the *Create Build Script* tool, the NI LabVIEW™ Application Builder (included in NI LabVIEW™ Professional Development System) is required.

Contents

1	Purpose.....	4
2	References	4
3	Introduction.....	4
4	Development distribution.....	5
4.1	Using prefixes	7
4.2	GOOP 2.0.....	7
5	Building executables containing GOOP 2.0.....	8
6	System requirements	9
7	Example	9
7.1	Development Distribution Tool example.....	10
7.2	Create Build Script Tool example.....	12

1 Purpose

The purpose of this document is to present the *Development Distribution Tool*. Explain why you need a development distribution tool and how to use it. It is assumed that the reader of this document has basic LabVIEW™ knowledge and also knows a bit about software engineering.

2 References

[1] GOOP Inheritance Toolkit – White Paper. Available at www.symbio.com

3 Introduction

When you have developed a LabVIEW™ application and needs to distribute and make a release of the application, usually one of the four strategies are used:

1. Just deliver the LabVIEW™ code as it is.
2. If the code is version controlled, release a version in the Source Code Control (SCC) tool¹ by applying labels and setup a baseline.
3. Create a LabVIEW™ LLB with “Save with options...” containing all VIs needed in the application and maybe apply a password or remove diagrams.
4. Build Application (.exe) with the NI LabVIEW™ Application Builder.

There may also be other ways and combination of the above strategies. Normally, the distributed application has some kind of version attach to the release. If you work in a SCC tool, you probably attach a *label* with the version. Usually versions are denoted as R2A, R3B etc or using a numeric denotation as 2.0, 6.1.2 etc. Other variants may also occur of course.

Larger application is usually build with *modules*, where each module has its own version. One of the major benefits of using modules is that they might be used in several applications and thereby be shared and reused by different applications. It will be robust against changes. When releasing an application distribution consisting of many different modules with versions, normally a *baseline* document is created. This document describes which modules and version of the modules that are needed for the specified release application. One baseline document is created for every new release. To specify a baseline, the SCC tool is used².

If LabVIEW™ code is supposed to be distributed to customers, the code has to be packaged some how. One option is to distribute the code as it is, with the same folder structure and a lot of VI. Sometimes that might be an option, but mostly you want an easy distribution, maybe just one single file containing all VIs, especially when the customer is not going to edit the code. Saving all needed VIs for the application in one large LLB solves this. An LLB is easy to copy and easy to install. Probably you want to apply a password to the diagrams as well if you want the LabVIEW™ code to

¹ ClearCase and SourceSafe are two examples of Source Code Control (SCC) tools.

² This is not explained here and is part of Configuration Management (CM). Please refer to a book about CM and SCC if you like to know more.

be non-editable, or maybe remove³ diagrams. This is the third strategy mentioned above.

For example, the Development Distribution Tool is not developed, as it looks when installed, in a large LLB with all VIs in one place. The source code is in a folder structure which is under SCC. Each time a new release is performed, labels are attached in the SCC and a baseline document is created with all the needed module versions. Then all needed VIs is saved into a large LLB. This illustrates that two of the strategies are used when releasing a new version of the tool.

However, distributing LabVIEW™ code as it is or creating a LLB containing all needed VIs requires that the users has LabVIEW™ installed which of course requires a LabVIEW™ license. To solve this problem, building an application (.exe) with an installer with the NI LabVIEW™ Application Builder⁴ may be used. This will install the LabVIEW™ Run-Time engine on the deployment machine and requires no LabVIEW™ installation or LabVIEW™ license for the user.

4 Development distribution

The *Development Distribution Tool* is used for distribution of LabVIEW™ code. It will *copy* the needed VIs, type def controls and menus into a distribution LLB or folder. Passwords may be applied or the diagram might be removed.

What is the difference between the “Save with options...” in LabVIEW™ and this new tool?

- **Adding prefix.** Each VI, type def control and menu in the distribution may be *prefixed*. This means that all the VIs, type def controls and menus in the distribution have a *unique* name. This way cross-linking problem between the original LabVIEW™ source code and distribution code is avoided!
- **LLB or folder.** The distribution may be saved to a specified folder instead of a LLB. The top-level VIs is stored in the specified folder and all sub-VIs is stored into a subfolder. If LLB is used for the distribution, it is possible to use an existing LLB as distribution LLB.
- **More than one top-level VI.** It is possible to have more than one top-level VI. This way distributed applications may share sub-VIs.
- **Top-Level VIs may be saved outside the distribution LLB.** This is useful when there is more than one top-level VI. If there is more than one top-level VI in an LLB⁵ and the LLB is “double-clicked” all top-level applications will start, which not always is wanted.
- **Exclude paths.** Any VIs with paths matching the exclude paths is not included in the distribution. Typically *vi.lib* is not in the distribution because the contents in the *vi.lib* are always installed with LabVIEW™. It also

³ Removing diagram makes it impossible to recompile LabVIEW™ code and makes it not possible to move distributions between different LabVIEW™ versions and platforms. A new distribution is needed for each LabVIEW™ version and platform if the diagrams are removed.

⁴ Add-on package to LabVIEW™ (included in the NI LabVIEW™ Professional Development System).

⁵ If the distributed LLB is placed in the LabVIEW\project folder, all top-level VIs in a LLB will be shown in the LabVIEW™ Tools menu. However, if the top-level VIs is outside the LLB they will be visible in the Tools menu as well. (Adding a “_” as the first character in the filename will make the VI invincible in the Tool menu).

possible to select “Only dynamic”. This will only exclude dynamic called GOOP2 methods matching the path. This is very useful if a distribution should exclude certain subclasses that will not be included in the distribution, but will include those VIs of the subclass that actually are used as sub-VIs by the top-level applications.

- **Dynamically called VIs.** Dynamically called VIs⁶ by a VI server in the application is included and also *all the sub-VIs needed*. (When using standard “Save with option...”, it is not possible to add dynamically called VIs. It is a tremendous task to manually copy these VIs into the LLB and also finds and copy all sub-VIs needed by the dynamic VIs.)
- **GOOP 2.0 dynamically called methods is included automatically.** Sub class methods called by virtual methods are included automatically in the distribution. See chapter 4.2.

These features above are NOT supported in the standard LabVIEW™ “Save with options...” The *Development Distribution Tool* with all of these new features *makes it really easy to distribute LabVIEW™ code* in a professional way.

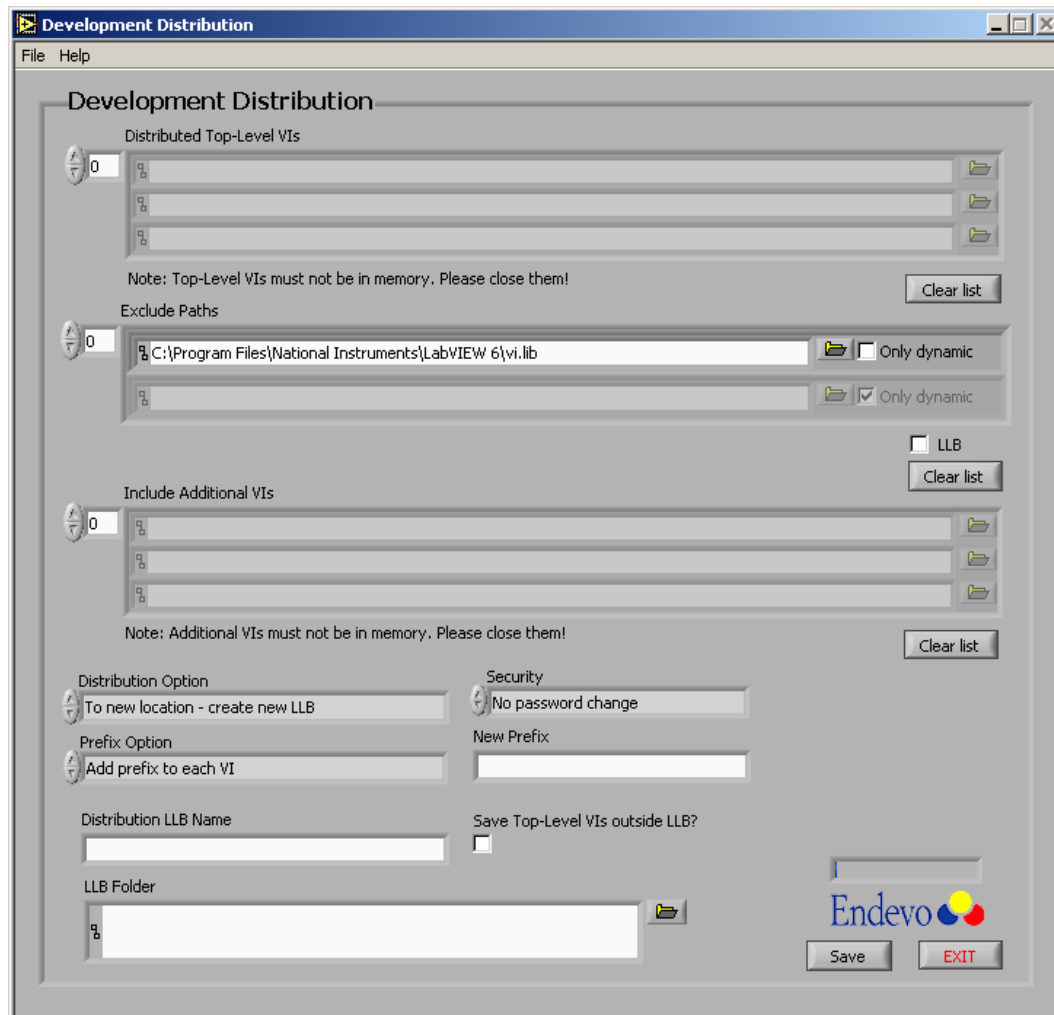


Figure 1. The front panel of the *Development Distribution Tool*.

⁶ Do not add virtually called GOOP2.0 methods, they will be included automatically, even though a VI server calls them, see ref [1].

4.1 Using prefixes

What is a prefix and why should I use this? A prefix is a string added at the beginning of the filename for all VIs, type def controls and menus.

Example: The prefix, “Test_R1A_” is used in a LabVIEW™ application that is going to be distributed. The application consists of three VIs, one main top-level VI and two sub-VIs:

MainApplication.vi
SubVI1.vi
SubVI2.vi

Now, using a prefix will give the following names in the distribution:

Test_R1A_MainApplication.vi
Test_R1A_SubVI1.vi
Test_R1A_SubVI2.vi

In a distribution LLB the Test_R1A_MainApplication.vi would be the top-level VI. However, all VIs is *renamed* in the distribution and the Development Distribution Tool will of course *relink* all VIs in the distribution to use the renamed VIs in the distribution.

What is this good for? Well, after the distribution is completed, there is no risk for an accidental cross-linking between the original LabVIEW™ source code and the distribution code! All names of the VIs have been changed and no cross-linking is possible. It is actually possible to run both the original source code at the same time as the distribution! This is really good if there is any problem⁷ with the distribution. Just open both the source code and the distribution at the same time and compare the behavior!

In this example, the **version was included in the prefix**. This is **strongly recommended**. This way it is really *easy to identify which version of the distribution code that is being used*. It is also possible to run two versions of the same distributed application at the same time!

4.2 GOOP 2.0

If you are not a GOOP 2.0 user, this chapter might be skipped (and also chapter 5). For more information about GOOP 2.0 and the GOOP Inheritance Toolkit⁸, please see ref [1]. To run an application containing GOOP 2.0 classes, make sure that the _GOOP2.llb is installed in the vi.lib\addons folder .

GOOP 2.0 uses VI Server for implementing the dynamic binding. Anyone that has tried to use the “Save as...” or Application Builder in LabVIEW™ knows that dynamically called VIs by a VI server needs to be included in the application

⁷ Problems with relative paths are quite common.

⁸ The Development Distribution Tool is included in the GOOP Inheritance Toolkit.

manually. Just building an application that contains a lot of virtual method that will dynamically call sub class methods will actually result in the sub class implementation *not being included* in the distribution. However, there is a solution to this problem. The ***Development Distribution Tool will automatically find and add dynamically called sub class method to the distributed application.***

There is one condition for the tool to work properly and find all dynamically called methods. **At least one VI from each sub class must be included in the LabVIEW™ source code.** Normally this is no problem, because the constructor method, *Create*, must be used somewhere to create the class and is always used somewhere in the code. The constructor of a sub class may be the only sub class method actually used in the application if only virtual base class methods are used.

Sometimes it is not wanted to distribute all subclasses in a distribution that never will be instantiated for the particular application, actually that will only cause the distribution to get larger than necessary. However, usually it is not a good idea to just exclude the class from the distribution, because that normally ends up that there are sub-VIs missing in the distribution (typically the *Create* method that is normally somewhere in the code as described above). By selecting “Only dynamic” when excluding a subclass, this will only exclude dynamically called methods and not needed sub-VIs. No VIs from that subclasses more than necessary will be included to make the distribution executable. Notice that the GOOP Wizard, Inspector and Objects In Memory tools will not work for distributed code.

5 Building executables containing GOOP 2.0

The NI LabVIEW™ Application Builder is used to deploy a LabVIEW™ application as an .exe file with an optional installer. This requires no LabVIEW™ license or LabVIEW™ installation on the deployment target machines. In the File menu of Development Distribution Tool, it is possible to start a support tool, called *Create Build Script*. This additional tool is used to create a build script (.bld file). Why is this needed? If the LabVIEW™ application to be deployed with the Application Builder, has any GOOP 2.0 classes that uses dynamic binding to virtually call dynamic methods, all of these methods must be added as dynamic VIs in the Application Builder. This might be quite a job and it is really easy to miss adding⁹ a sub class method and thereby changes the behavior of the deployed application.

However there is an easy solution to this. *The Create Build Script Tool analyzes an application and finds all dynamically called sub class method¹⁰.* The tool creates a build script that might be loaded into the LabVIEW™ Build Application where all dynamically called subclass methods found are added as dynamic VIs.

To deploy a LabVIEW™ application containing GOOP2.0 classes with virtually called sub class methods perform the following steps:

⁹ This will be the same as if the sub class do not implement the method and the base class implementation of the method will be used instead.

¹⁰ As explained in chapter 4.2, at least one VI from each sub class must be present in the source code. Normally, the constructor, *Create*, for each sub class is included somewhere in the source code.

1. Run the Create Build Script Tool to analyzes and create a build script (.bld file). All possible called sub class methods are added as “Dynamic VIs”.
2. Start the NI LabVIEW™ Application Builder.
3. Load the build script created in 1.
4. Configure the Build Application Tool as you please and build the application.

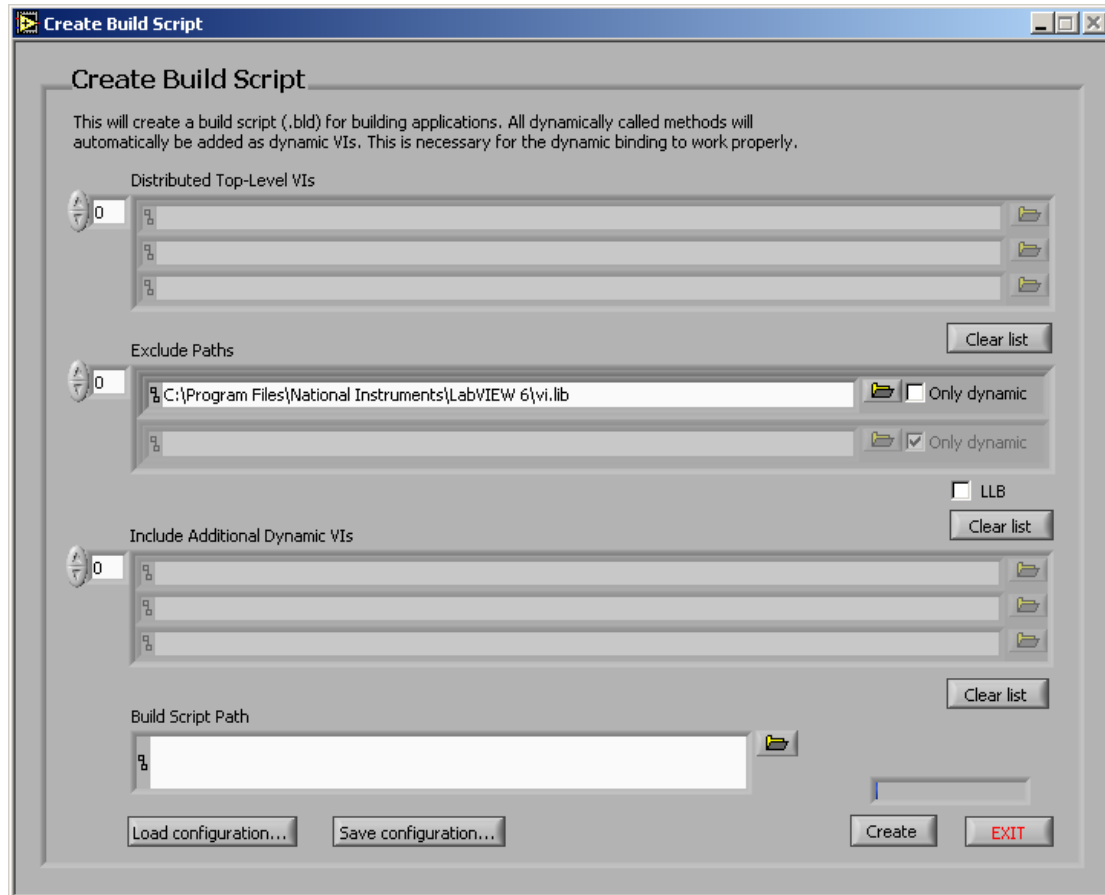


Figure 2. The front panel of the *Create Build Script Tool*. It creates a build script for an application containing GOOP2.0 classes with virtually called sub class methods automatically added as “Dynamic VIs” in the build script.

6 System requirements

- LabVIEW™ 6.0.2 or later (all development system options)
- For building applications (.exe), the add-on package NI LabVIEW™ Application Builder is required (included in NI LabVIEW™ Professional Development System)

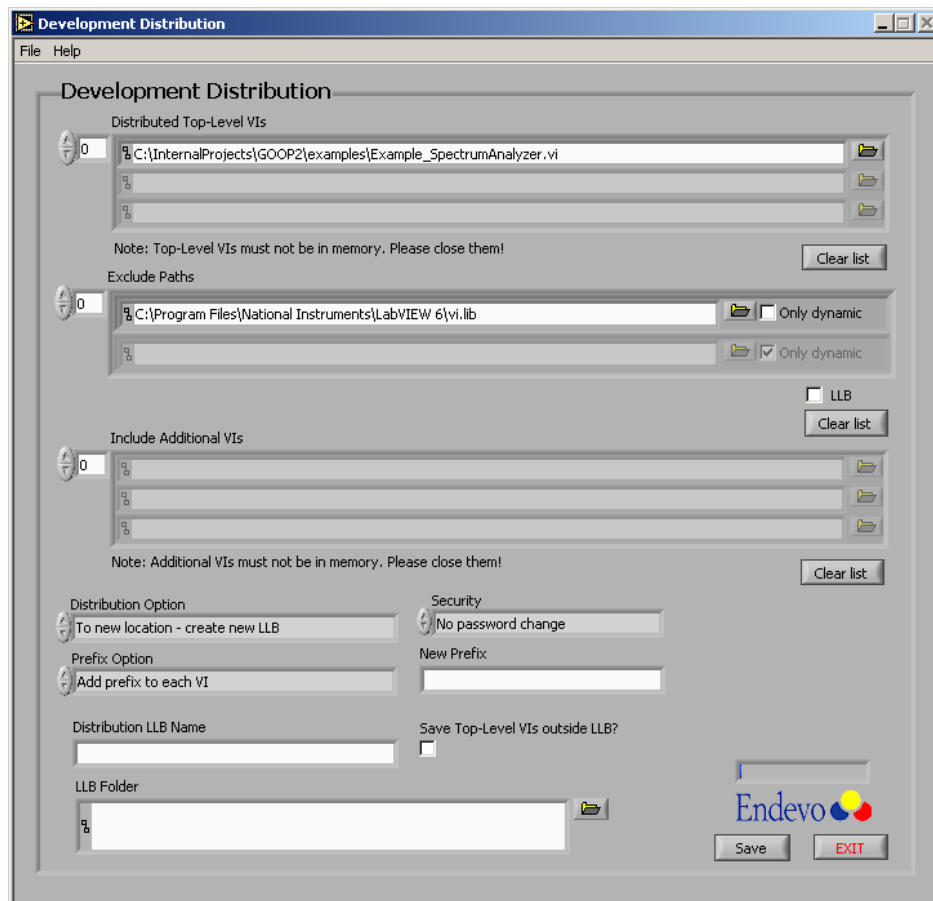
7 Example

The first example shows a LabVIEW™ application to be distributed with the *Development Distribution Tool* as an LLB, where all VIs are prefixed and a password is applied. It is actually the same application used as an example in ref [1], GOOP Inheritance Toolkit – White paper.

In the second example, the same LabVIEW™ application is going to be deployed as an .exe file using the *Create Build Script Tool* to analyze and create the build script. After this, the script is loaded into the LabVIEW™ Application Builder to create the .exe file.

7.1 Development Distribution Tool example

1. Start the Development Distribution Tool¹¹ from the LabVIEW™ Tools menu.



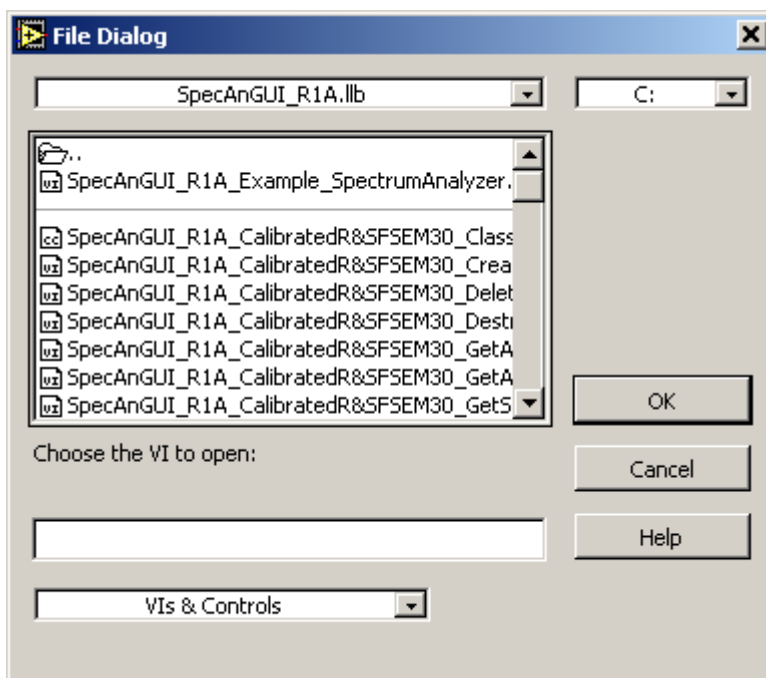
2. Select the main top-VI to be distributed.
3. Select the distribution option - “To new location – create new LLB”
4. Select prefix option – “Add prefix to each VI”
5. Write the new prefix – in this example “SpecAnGUI_R1A_” (both the application name and version is included in the prefix)
6. Select LLB name – in this example “SpecAnGUI_R1A” (almost identical to the prefix).
7. Select LLB folder – the folder where the LLB should be created.

¹¹ It is also possible to start the *Development Distribution Tool* from the GOOP Wizard 3.0 File menu.

8. Press “Save”.
9. A password dialog prompts. Enter the new password to be applied and also confirm the password. If the LabVIEW™ source code is already password protected, these “old” passwords might be entered in the “Old Passwords” list. If not, the new password for the already password protected VIs cannot be applied.

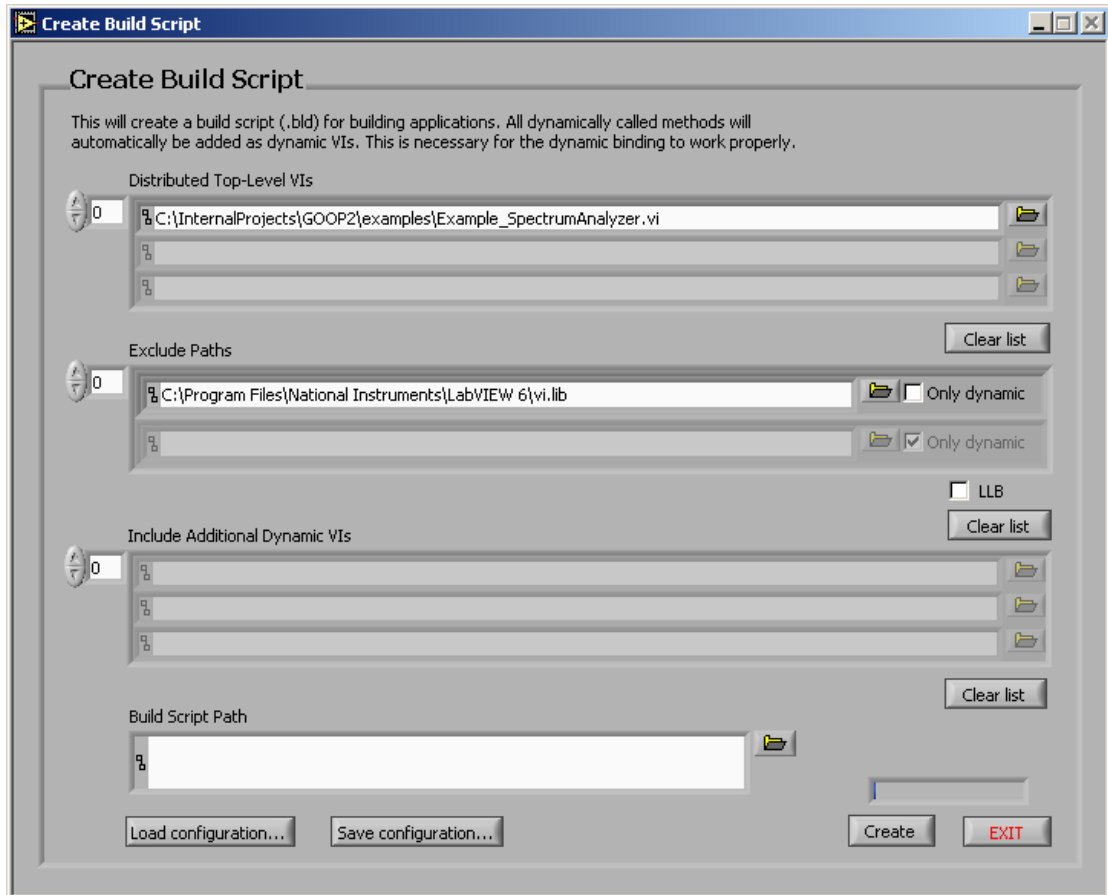


10. Press “Continue”! Notice that it might take a while to analyze a large application.
11. Finished! All VIs, type def controls and menus needed are now in the distribution LLB with a prefixed name. Make sure that the _GOOP2.llb is installed in the vi.lib\addons folder if the distribution contains any GOOP 2.0 classes.



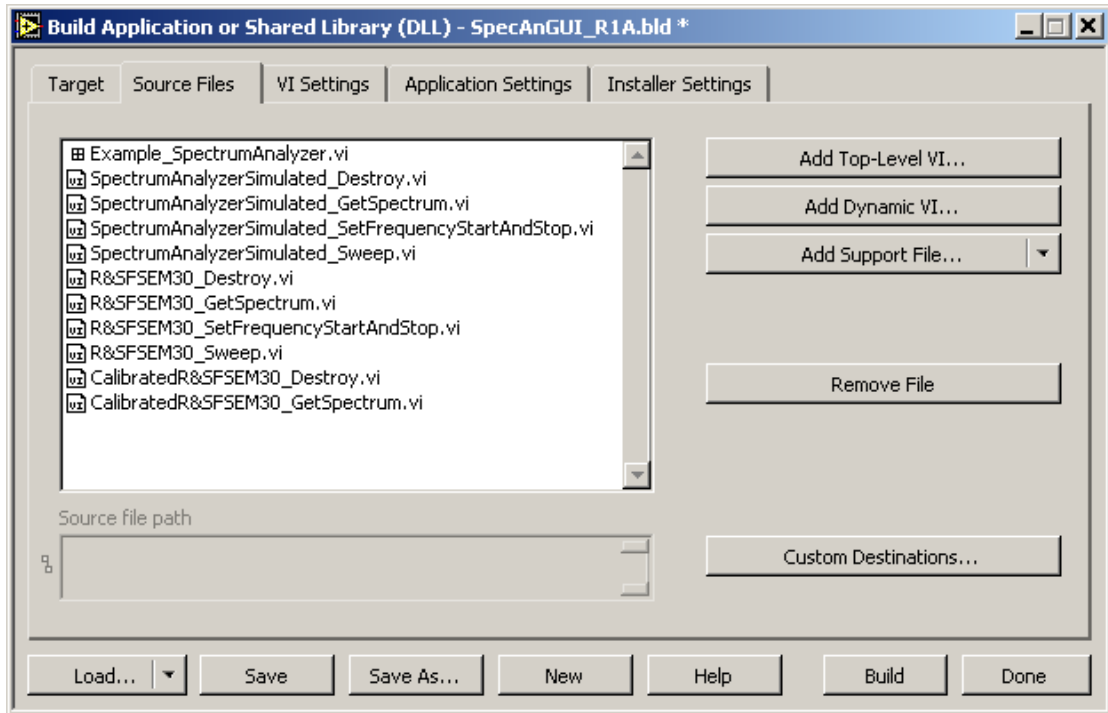
7.2 Create Build Script Tool example

1. Start the *Create Build Script Tool*¹² from the Development Distribution Tool File menu.



2. Select the main top-level VI to be analyzed and later deployed as an .exe file.
3. Select a name and path to the .bld build script file.
4. Press “Create”. Notice that it might take a while to analyze a large application.
5. Start the LabVIEW™ Application Builder from the LabVIEW™ Tools menu -> “Build Application or Shared Library (DLL)”.
6. Press “Load...” and select the create build script (.bld). Notice how all virtually called sub class methods automatically is added as “Dynamic VI” on the “Source Files” tab.

¹² It is also possible to start the *Create Build Script Tool* from the GOOP Wizard 3.0 File menu.



7. Configure the application builder as you normally would.

8. Make the build! Finished!