Getting Started with the JNBridgePro Plug-ins for Visual Studio and Eclipse

Version 12.0

jnbridge.com

JNBridge, LLC
jnbridge.com

April 7, 2025

Table of Contents

Introduction

This document shows how to use the JNBridgePro plug-ins for Visual Studio and for Eclipse to generate proxies and to use them in larger projects. Users can generate their proxies within Visual Studio 2017 and 2019, and within Eclipse 3.2 through 4.10, and use the generated proxies seamlessly in their project builds.

This document assumes that the users are familiar with the JNBridgePro standalone proxy generation tool, and with how the generated proxies can be used. For more information on these topics, please see the examples that ship with the JNBridgePro installation, and also the *Users' Guide*.

JNBridgePro plug-in for Visual Studio

The JNBridgePro plug-in for Visual Studio can be used with Visual Studio 2017, and 2019, and is used in projects where .NET code is calling Java code. The example is taken from the "log demo" that comes with the JNBridgePro installation. For more information on the log demo, please see the document associated with it that comes with the installation

Installing the Visual Studio plug-in (Visual Studio 2017 or 2019)

Up through Visual Studio 2015, the Visual Studio plug-in is installed automatically when JNBridgePro is installed in development mode. Starting with Visual Studio 2017, an additional step must be performed. In the JNBridgePro installation folder, locate the folder "VS plugin (2017 and later)." In that folder, you will find a file JNBridgePlugin2017.vsix. Assuming Visual Studio 2017 and/or 2019 is installed, double-click on this file in Windows Explorer. The Visual Studio extension manager will come up. When prompted, select the versions of Visual Studio in which you want the plug-in to be installed, then follow the remaining instructions. When the process is complete, the plug-in will be installed.

The plug-in packaged in the VSIX file will only work if JNBridgePro has already been installed in development mode *on that machine*, and if a valid development license has been deployed to that machine.

Generating the proxies

Start by creating your new solution, and a C# console application. Add the files App.config and LoggerDemo.cs (Figure 1).

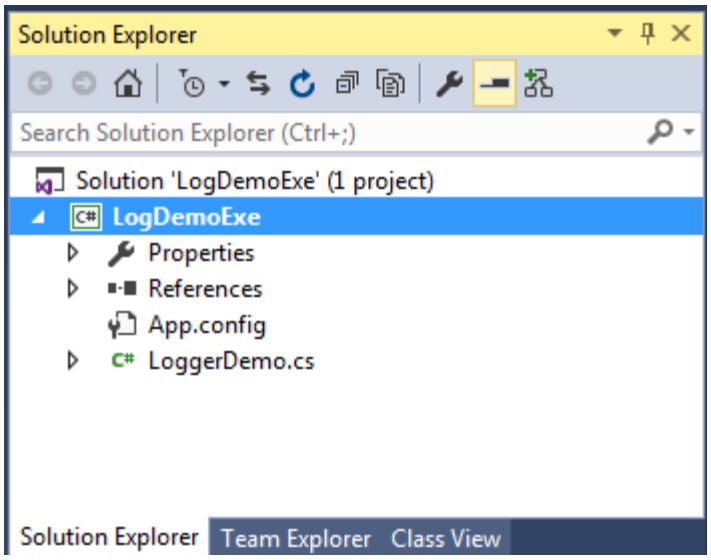

Figure 1. Log demo project

Next, create a new proxy generation project. There are several ways to do this, but the simplest way is to right-click on the solution node in the Solution Explorer, then select **Add→New Project...**. In the Add New Project dialog box that now appears, note that there is a new project type, JNBridge, and a new template, DotNetToJavaProxies. Select that template, then name the new project and assign it a location (Figure 2 and Figure 3).

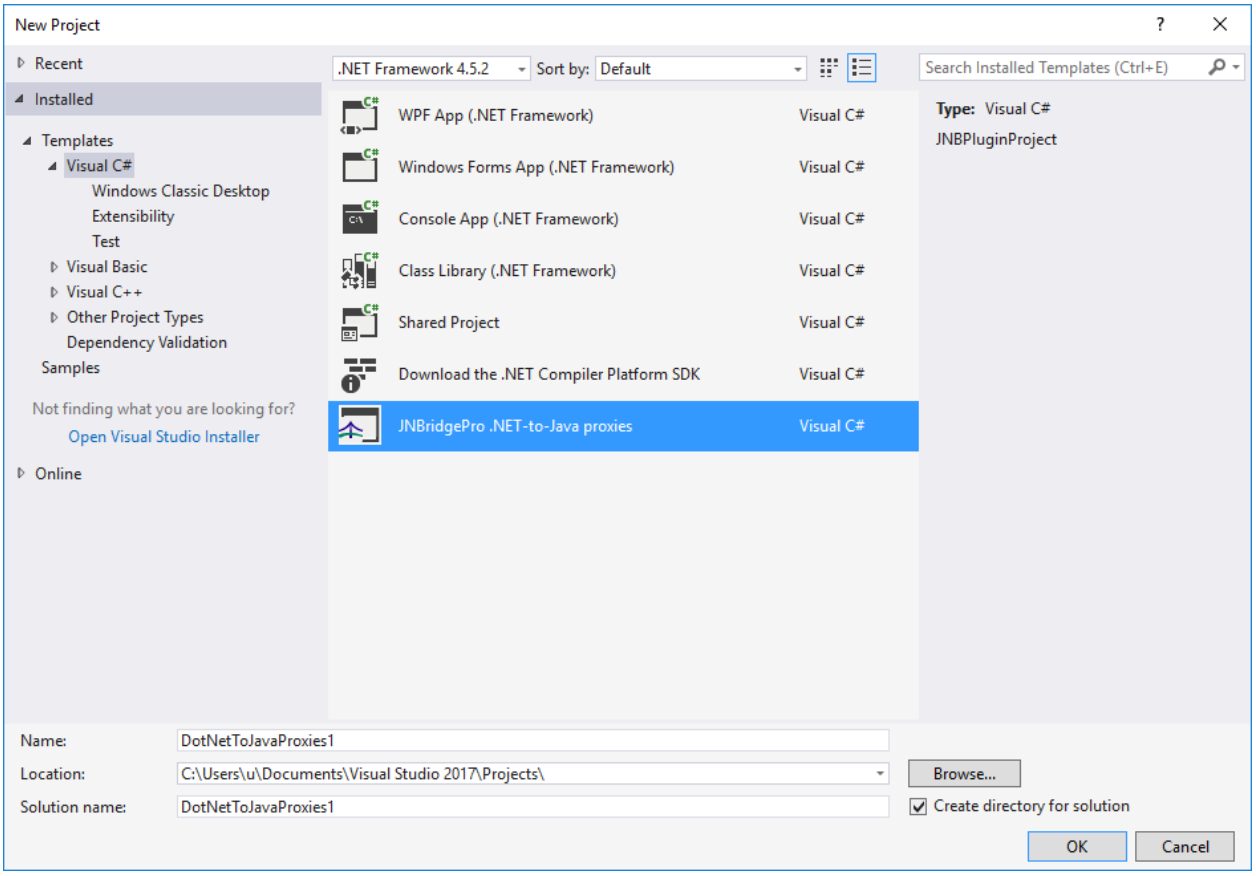

Figure 2. Adding a new JNBridge proxy generation project (VS 2017)

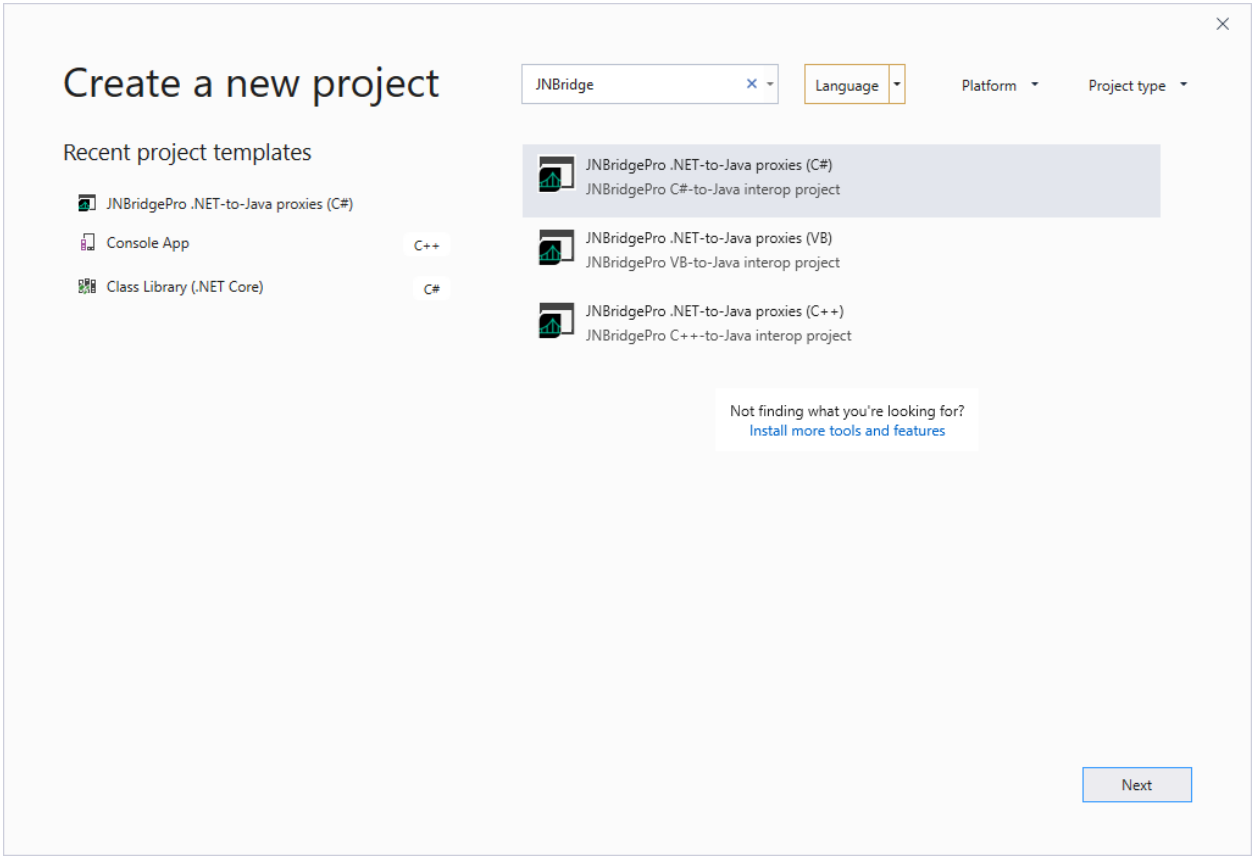

Figure 3. Adding a new JNBridge proxy generation project (VS 2019)

Note that the Solution Explorer now contains a new proxy generation project, and a new proxy generation document, a .jnb file. This is the same .jnb file used by the standalone proxy generator (Figure 4).

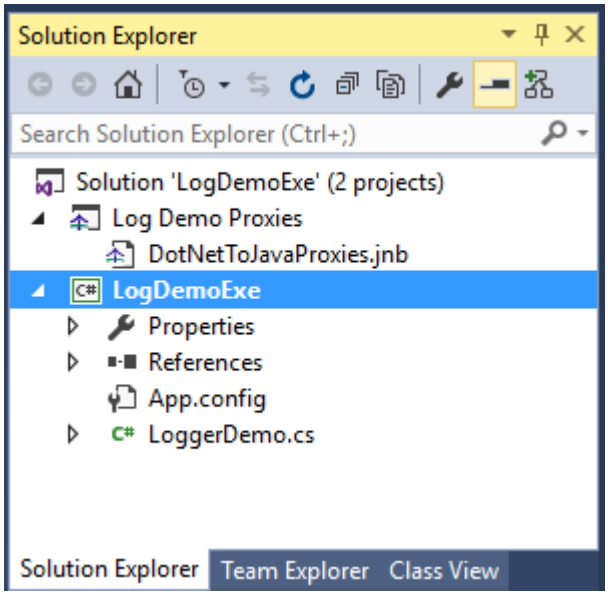

Figure 4. After adding the proxy generation project

Open the .jnb file by double-clicking on its node in the Solution Explorer. An editor window will open in Visual Studio. Note that its layout resembles the GUI version of the standalone JNBridgePro proxy generation tool (Figure 5).

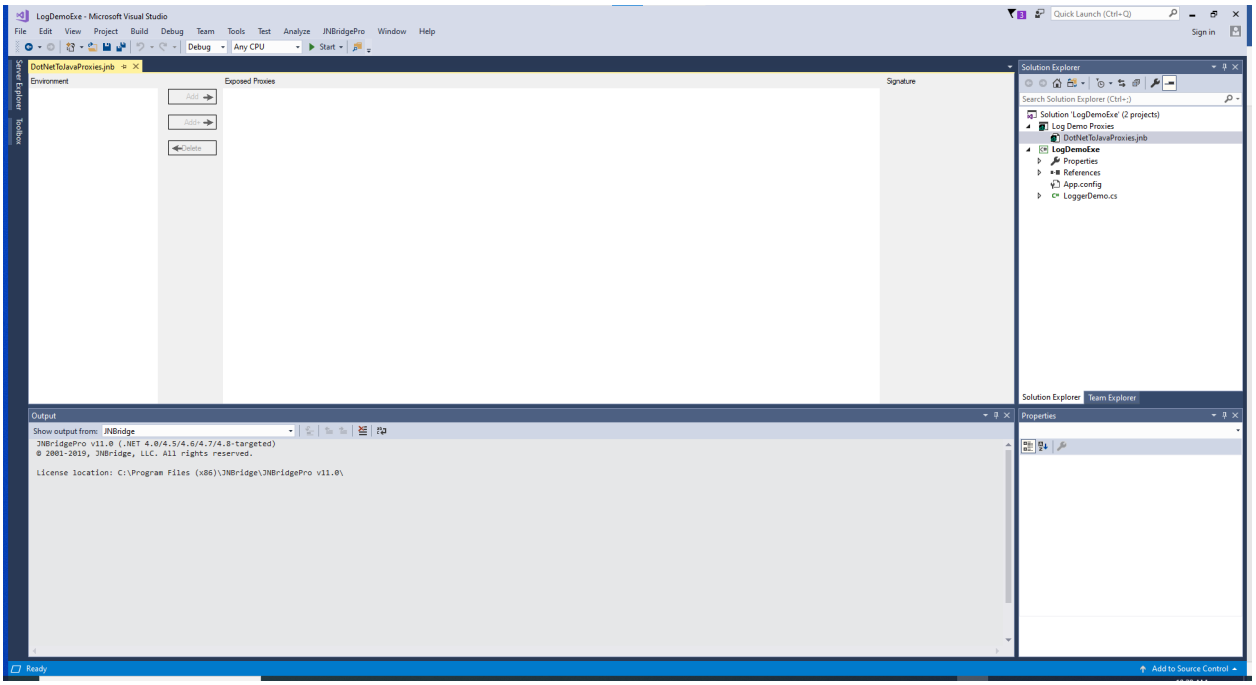

Figure 5. After opening the .jnb file

Next, add the files log4j.jar and log4j-core.jar to the class path to be searched for proxy generation. (You can download the log4j JAR files from http://jakarta.apache.org/log4j/docs/index.html.) Also add the folder in which loggerDemo\JavaClass.class is to be found. Use the menu command

JNBridgePro→Edit Classpath.... (Alternatively, you can right-click on the .jnb file node in the Solution Explorer and select **Edit Classpath...** or use the Edit Classpath button in the JNBridgePro toolbar.) The **Edit Class Path** dialog box will come up, and clicking on the **Add...** button will bring up a dialog that will allow the user to indicate the paths of the Jar and class files (Figure 6).

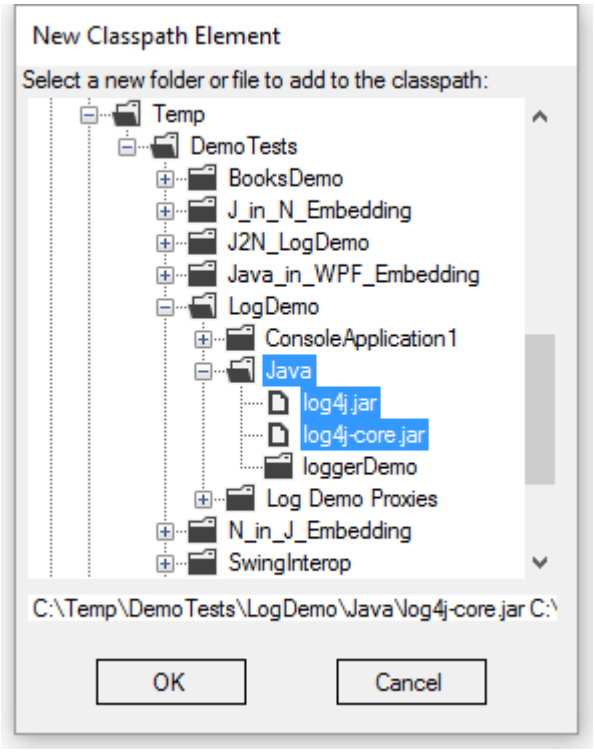

Figure 6. Adding a new classpath element

When all the necessary elements of the classpath are added, the **Edit Class Path** dialog should contain information similar to that shown in Figure 7.

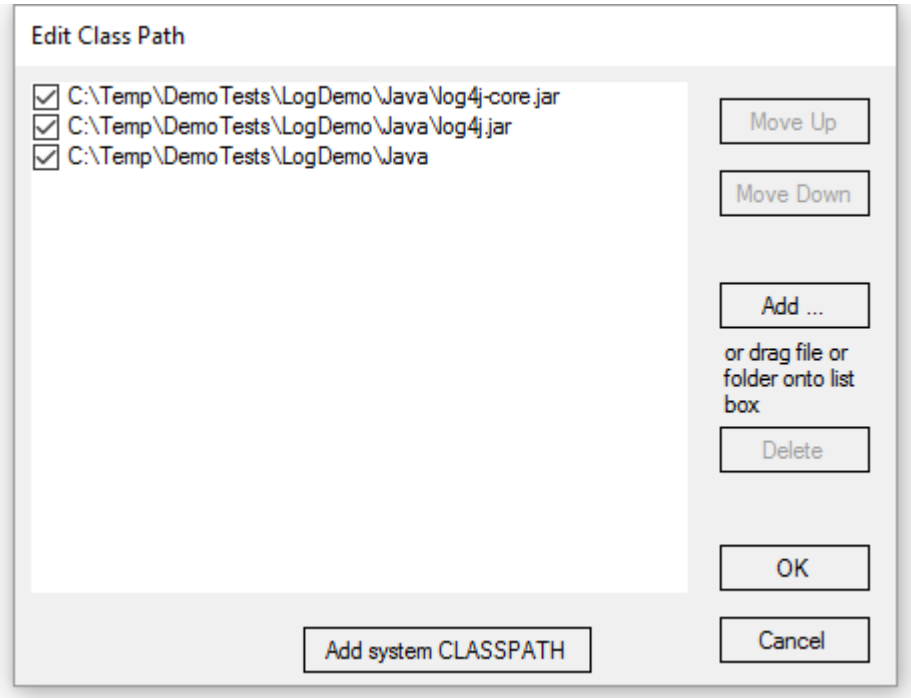

Figure 7. After creating classpath

The next step is to load the classes from each of the Jar files, and to add JavaClass. For the Jar files, use the menu command **JNBridgePro→Add Classes from JAR File…** for each Jar file. For a single class such as JavaClass, use the menu command **JNBridgePro→Add Classes from Classpath…** and enter the fully qualified class name loggerDemo.JavaClass (Figure 8). (You can also accomplish these actions by right-clicking on the .jnb file node in the Solution Explorer, or by clicking on the appropriate button in the JNBridgePro toolbar.)

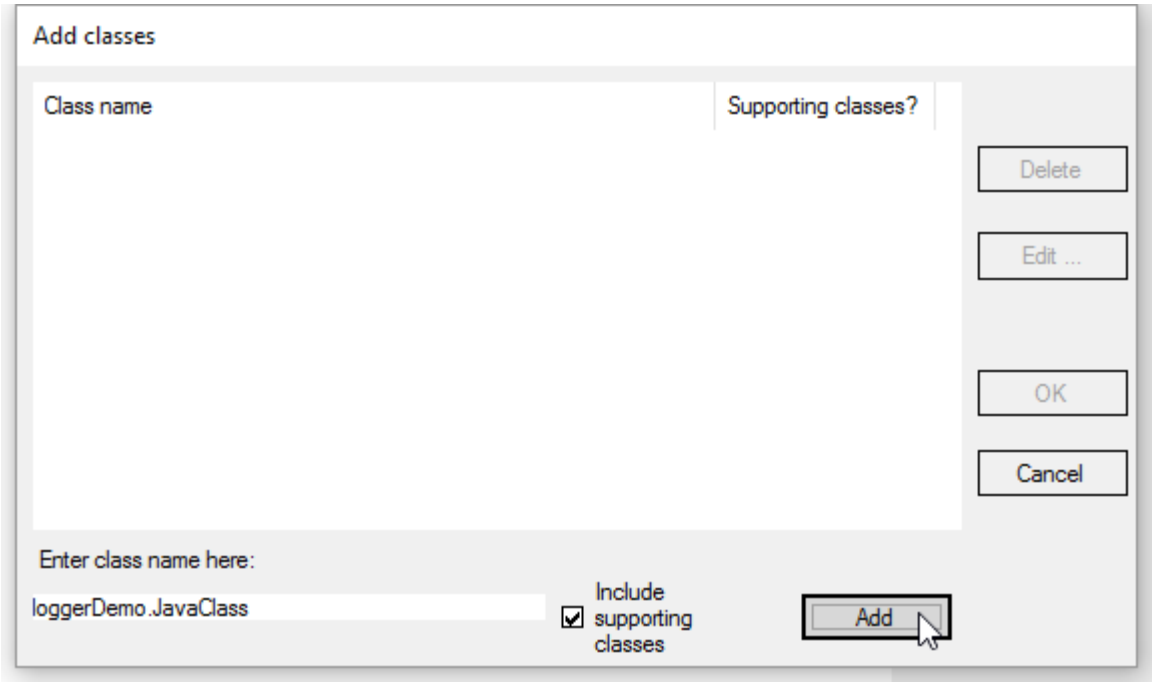

Figure 8. Adding a class from the classpath

Loading the classes may take a few minutes. Progress will be shown in the output window (in the JNBridge sub-pane) in Visual Studio, and in the progress bar. When completed, the classes in the log4j Jar files and loggerDemo.JavaClass will be displayed in the Environment pane on the upper left of the editor (Figure 9). Note that JNBridgePro will warn us that we are missing a number of classes relating to JMS (Java Messaging Service), XML, and JavaMail. Since we are not going to use these capabilities of log4j, we can safely ignore this warning.

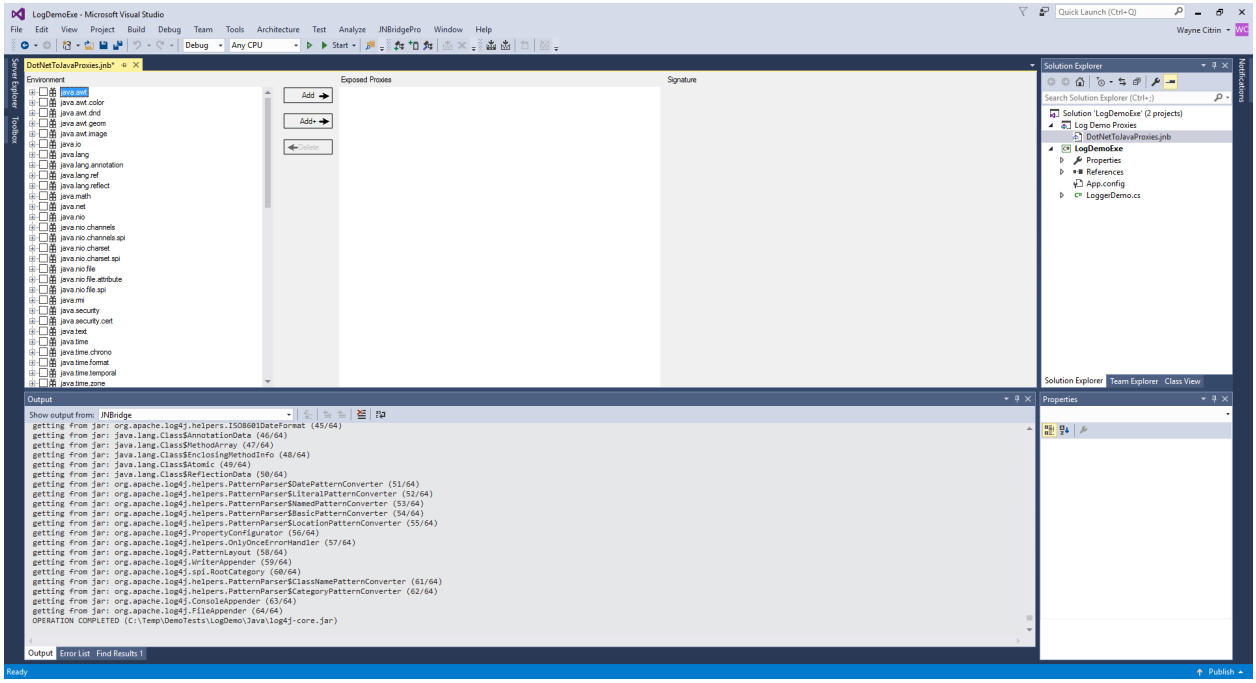

Figure 9. After adding classes

We wish to generate proxies for all these classes, so when all the classes have been loaded into the environment, make sure that each class in the tree view has a check mark next to it. Quick ways to do this include clicking on the check box next to each package name, or simply by selecting the menu command **JNBridgePro→Check All in Environment**. Once each class has been checked, click on the **Add** button to add each checked class to the list of proxies to be exposed. These will be shown in the Exposed Proxies pane (Figure 10).

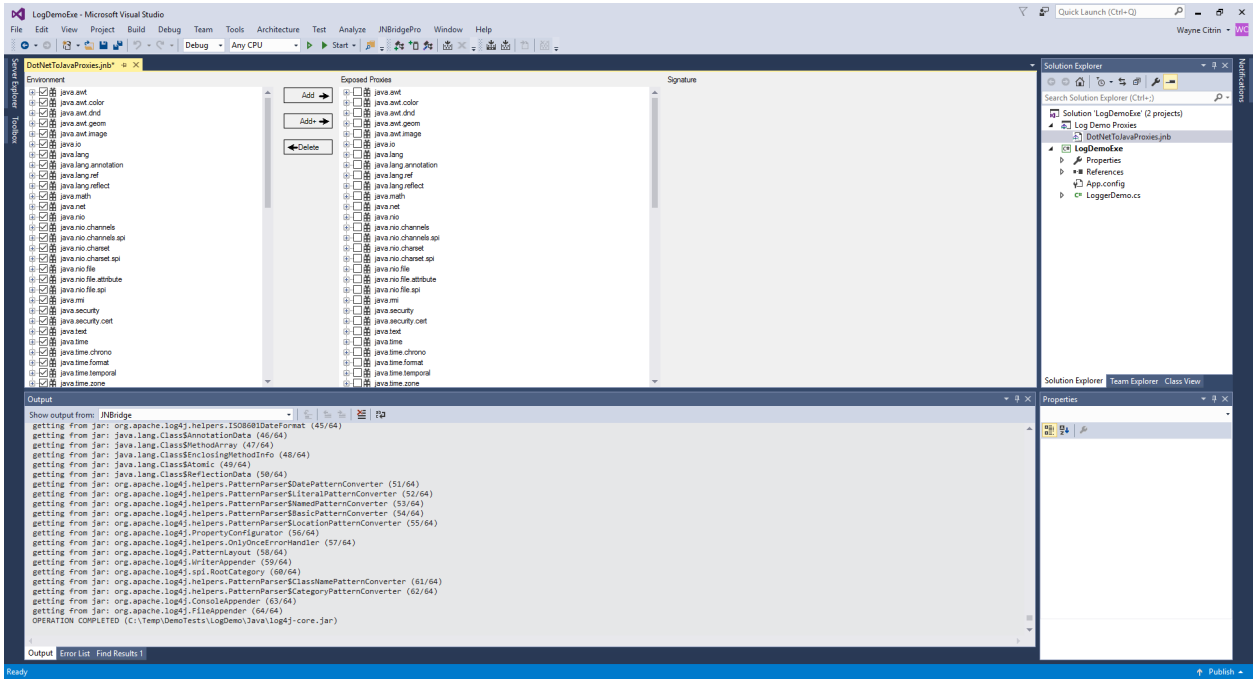

Figure 10. After adding classes to Exposed Proxies pane

While one can build the proxies now, using the JNBridgePro→Build menu item, in this example, we will wait until we have referenced the proxy generation project in the main executable's project and do the build as part of a build of the entire solution.

Using the proxies

Now that we have created the proxy generation project, we can reference the project from our main project by right-clicking on the project node for our main executable and selecting **Add Reference…**. In the Add Reference dialog box, select the Projects tab, and select the proxy generation project. Also, under the Browse tab, select jnbshare.dll and jnbsharedmem_x86.dll or jnbsharedmem_x64.dll (or both) from the JNBridgePro installation folder, using the 4.8-targeted versions and add them to the references (Figure 11).

Finally, add the licensing dlls jnbauth_x86.dll or jnbauth_x64.dll (or both) to the project. They're not .NET dlls, so you can't add them as references – instead, add them to the project through Add→Existing Item…, and make sure their Copy to Output Directory properties are both set to Copy Always.

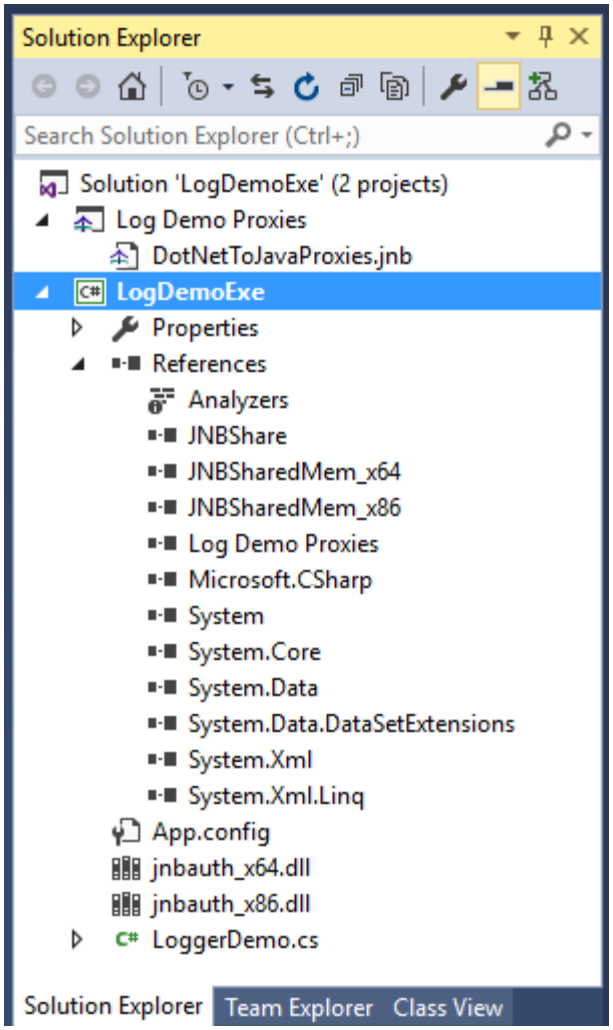

Figure 11. After referencing the proxy generation project

Once that's done, just build your solution. That's all there is to it. The output of the proxy generation project will be automatically used in your main executable's project.

Once you've built the proxies, they will appear in Visual Studio's IntelliSense when you program against them. (If you haven't yet built the proxy project, they won't appear in IntelliSense until you do.)

At this point, create, configure, and run your project as described in the "Log Demo" document.

JNBridgePro plug-in for Eclipse

The JNBridgePro plug-in for Eclipse can be used with Eclipse 3.2 through 4.13, and is used in projects where Java code is calling .NET code. The example is taken from the "Java-to-.NET" that comes with the JNBridgePro installation. For more information on the Java-to-.NET demo, please see the document associated with it that comes with the installation.

Before using the Eclipse plug-in, make sure it has been installed. Locate the file jnbridgepro11_0_0-eclipse.zip in the "Eclipse plug-in" folder located in the JNBridgePro 11.0 installation folder. Open the zip file and extract its contents (a folder "com.jnbridge.plugin.eclipse_2.4.0") to the plugins folder inside your Eclipse installation.

If you are using the 64-bit version of Eclipse, you must use the 64-bit version of JNBridgePro, and you must add the following argument to eclipse.ini:

```
-vm C:/Program Files/Java/jre8/bin/javaw.exe
```

or use a path to some other 64-bit javaw.exe. *Do not* use the path to the javaw.exe that resides in \Windows\System32.

Generating the proxies

Start by creating your main Java project. Here we will simply create a project and import the Java files supplied in the Java-to-.NET demo (found in the file winFormDemo22.zip in the JNBridgePro installation). Note that there are compilation errors because they reference proxy classes that do not yet exist (Figure 13).

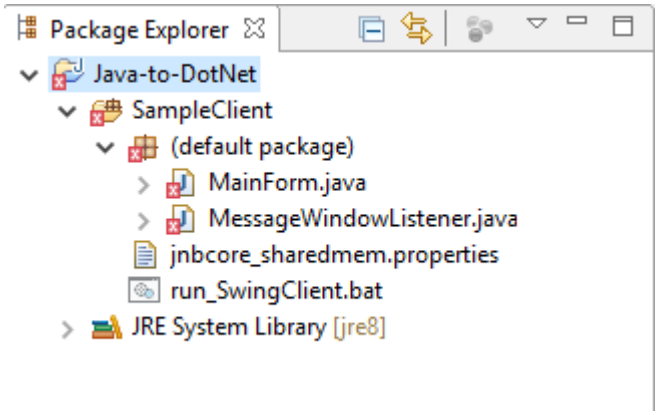

Figure 12. The initial project

Next, we will create our proxy generation project. Select the **File→New→Other…** menu item. The New dialog box will appear. Note that there are two new items under the JNBridge header: "Java to .NET Interoperability Project," and "Java-to-.NET Proxies" (Figure 14).

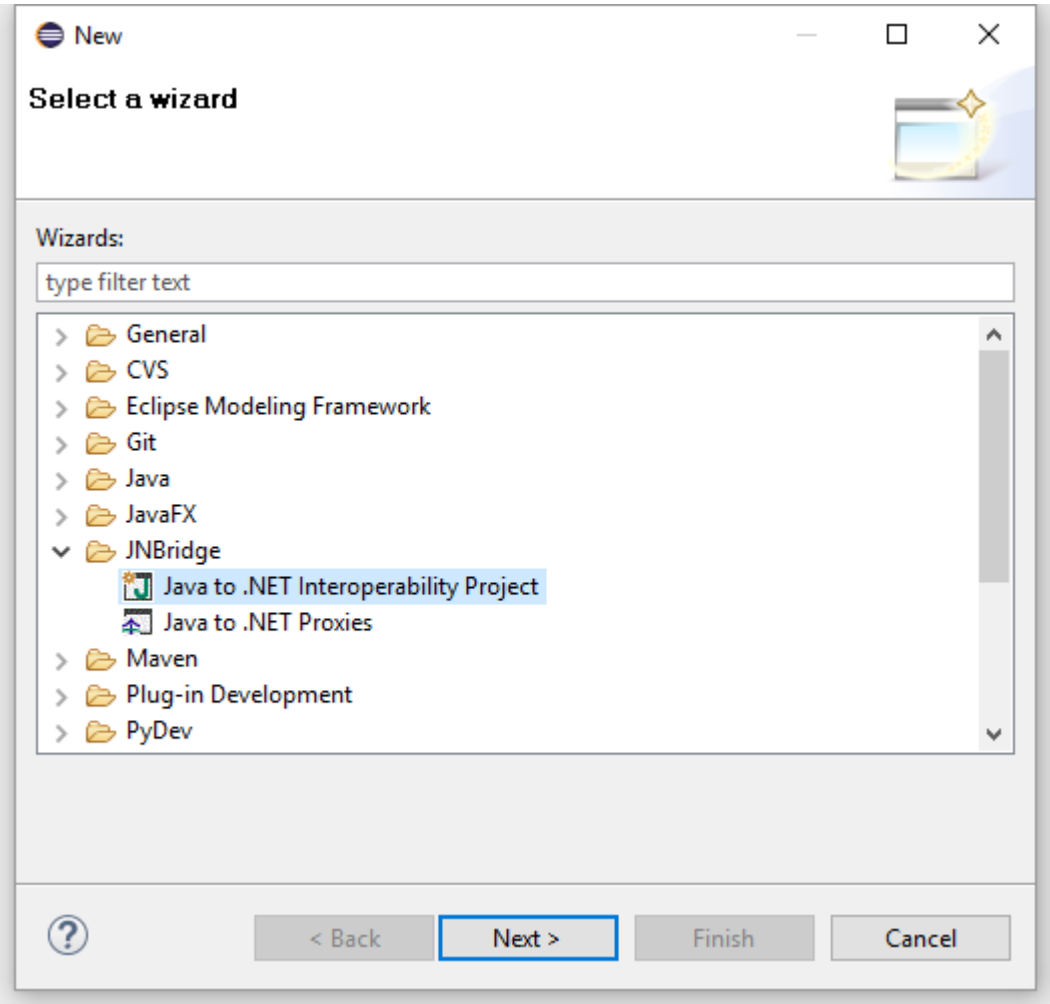

Figure 13. Creating a new interoperability project

Select “Java to .NET Interoperability Project” and click on the Next button. Give the project a name, and indicate its location. Do not reference any other projects.

Next, select the project node for the proxy generation. Right-click on the project node and select **New→Other…**. In the New dialog box this time, select “Java to .NET proxies” and click on the Next button. Again, select a name, but leave the location the same. The Package Explorer will now display the new proxy generation project (Figure 15). Inside the project will be a .jnb file node. This represents the same .jnb file that is used by the GUI-based standalone proxy generation tool.

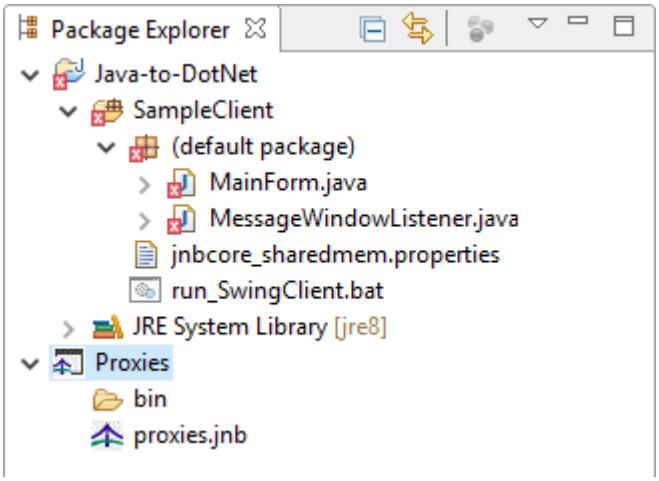

Figure 14. After adding the new proxy generation project

At this point, an editor window for the .jnb file should be displayed in Eclipse (Figure 16). If it is not displayed, you can display it by double-clicking on the .jnb file node in the Package Explorer.

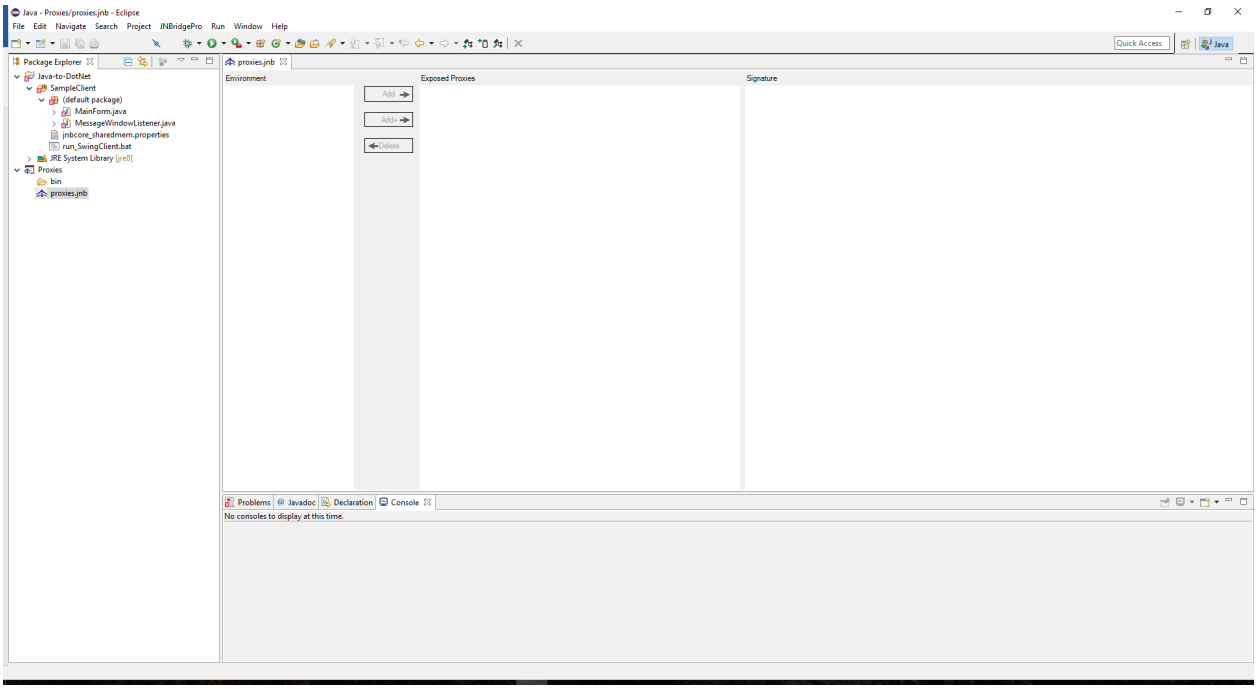

Figure 15. The proxy generation editor in Eclipse

Next, add the assemblies SwingInterop.dll and System.Windows.Forms.dll to the assembly list to be searched by JNBProxy. (We will be calling methods that are not defined in SwingInterop.Form1 and Form2, but rather in their superclass System.Windows.Forms.Form, which is defined in System.Windows.Forms.dll.) Use the menu command **JNBridgePro→Edit Assembly List...**. (Make sure that the proxy generation editor is active.) The **Edit Assembly List** dialog box will come up, and clicking on the **Add...** button will bring up a dialog that will allow the user to indicate the paths of

SwingInterop.dll (Figure 17). Alternatively, you can right-click on the .jnb file node in the Package Explorer and select **Edit Assembly List...**.

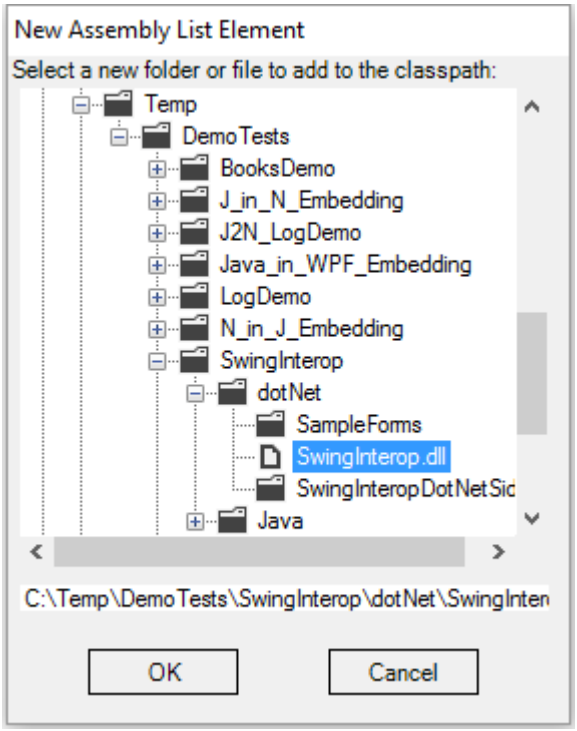

Figure 16. Adding a new assembly list element

System.Windows.Forms.dll is in the Global Assembly Cache (GAC). Add it to the assembly list by clicking on the **Add From GAC...** button, and selecting the System.Windows.Forms.dll from the displayed list (Figure 18). If you have more than one version of the .NET Framework installed on your machine, you may have more than one version of System.Windows.Forms.dll in the GAC; select the appropriate one.

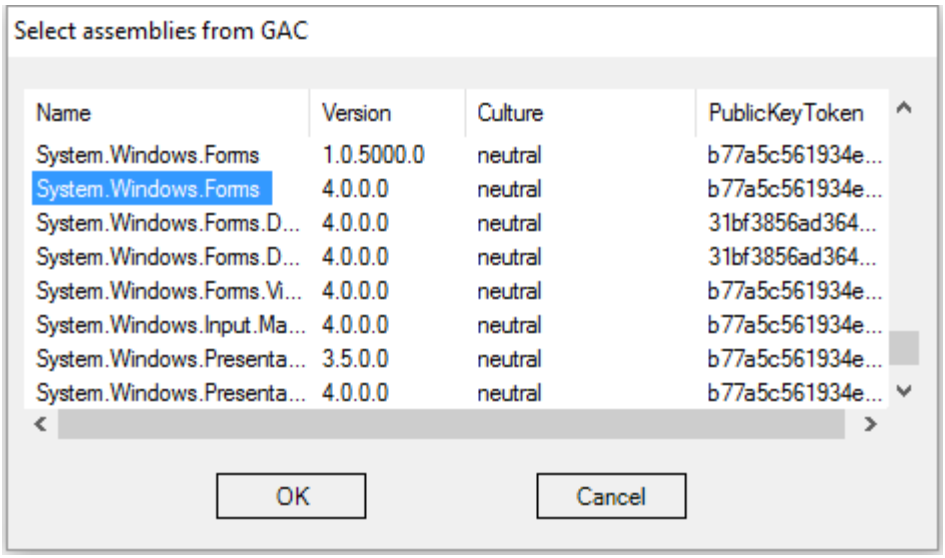

Figure 17. Selecting an assembly from the GAC

Note that we are using the .NET 4.0 version of System.Windows.Forms.

Figure 18. Targeting .NET Framework 4.0

When all the necessary elements of the classpath are added, the **Edit Assembly List** dialog should contain information similar to that shown in Figure 20.

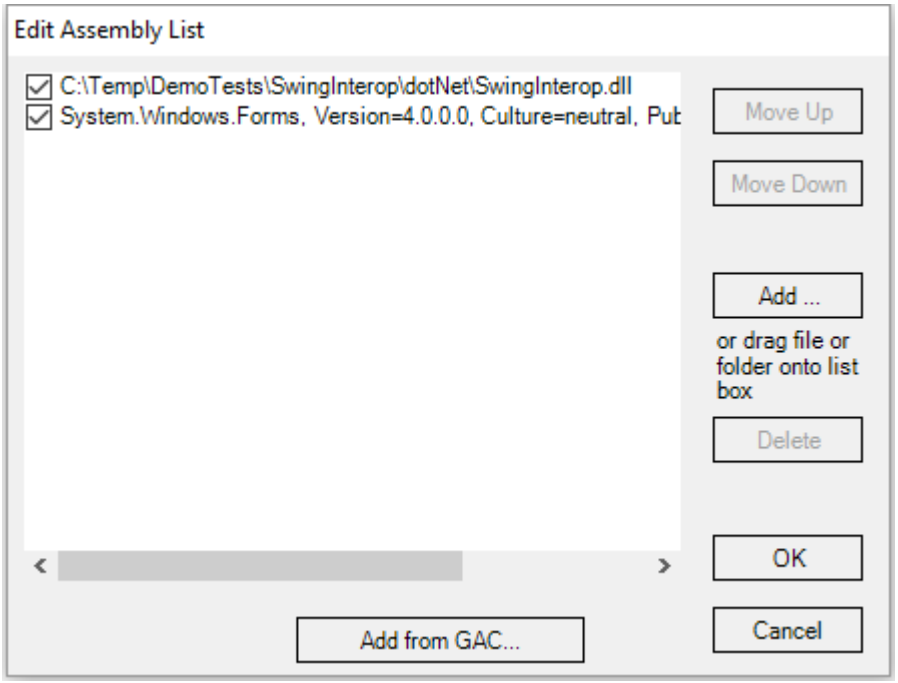

Figure 19. After creating assembly list

The next step is to load the classes Form1, Form2, and JavaWindowEventArgs, plus the supporting classes. Use the menu command **JNBridgePro→Add Classes from Assembly List...** and enter the fully qualified class names SwingInterop.Form1, SwingInterop.Form2, and SwingInterop.JavaWindowEventArgs, making sure that the "Include supporting classes" checkbox is checked for each (Figure 21). Alternatively, you can also right-click on the .jnb file node in the Package Explorer and select **Add Classes from Assembly List...**.

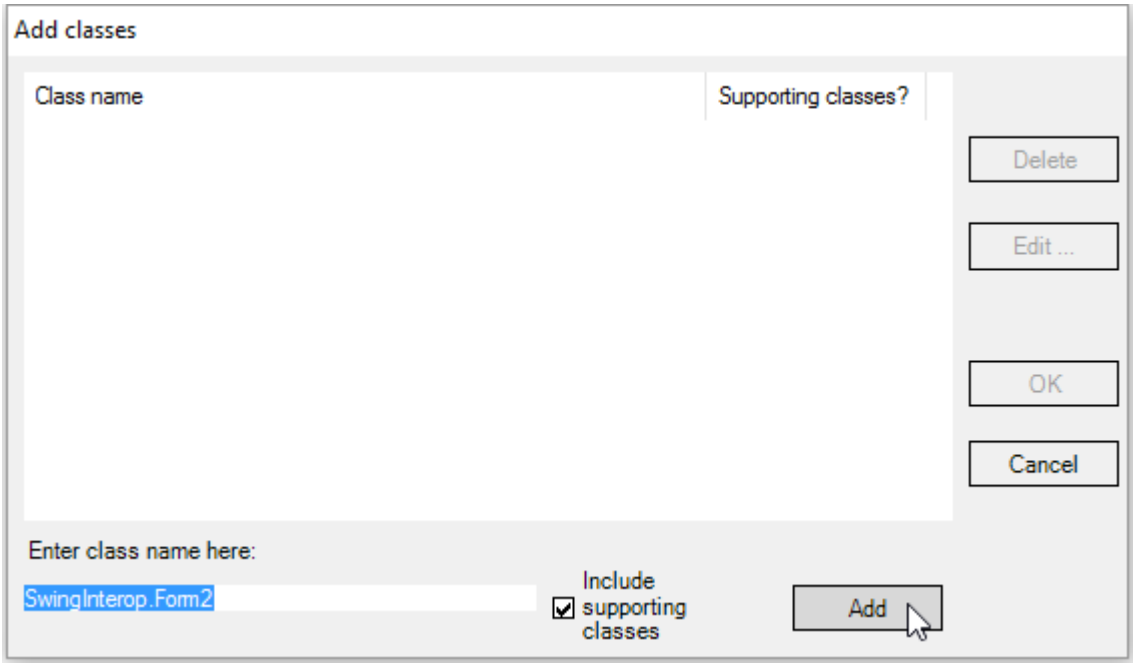

Figure 20. Adding a class from the assembly list

Loading the classes may take a few seconds. Progress will be shown in the console pane in the bottom of the window, and in the progress bar. (If you don't see a console, select the **Window→Show View→Console** menu item.) When completed, Form1, Form2, and all their supporting classes will be displayed in the Environment pane on the upper left of the proxy generation editor (Figure 22). Note that JNBridgePro will warn us that we are missing a number of classes. Since we are not going to use these capabilities, we can safely ignore this warning.

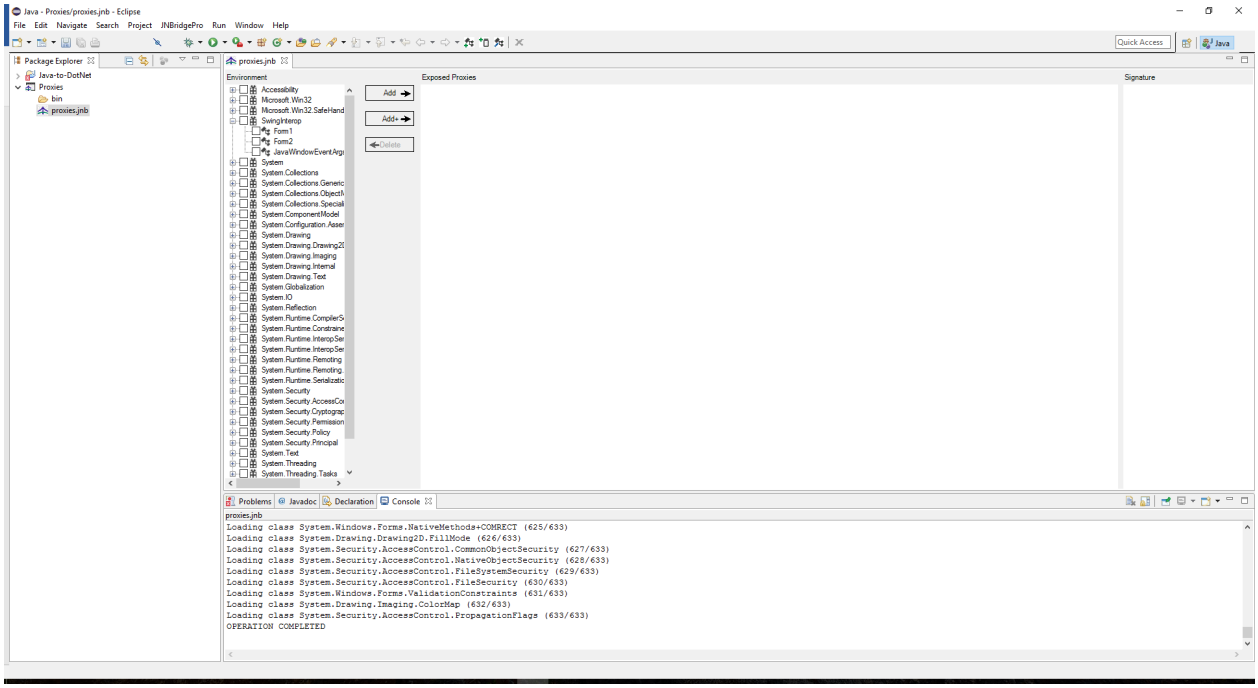

Figure 21. After adding classes

We wish to generate proxies for all these classes, so when all the classes have been loaded into the environment, make sure that each class in the tree view has a check mark next to it. Quick ways to do this include clicking on the check box next to each package name, or simply by selecting the menu command **JNBridgePro→Check All in Environment**. Once each class has been checked, click on the **Add** button to add each checked class to the list of proxies to be exposed. These will be shown in the Exposed Proxies pane (Figure 23).

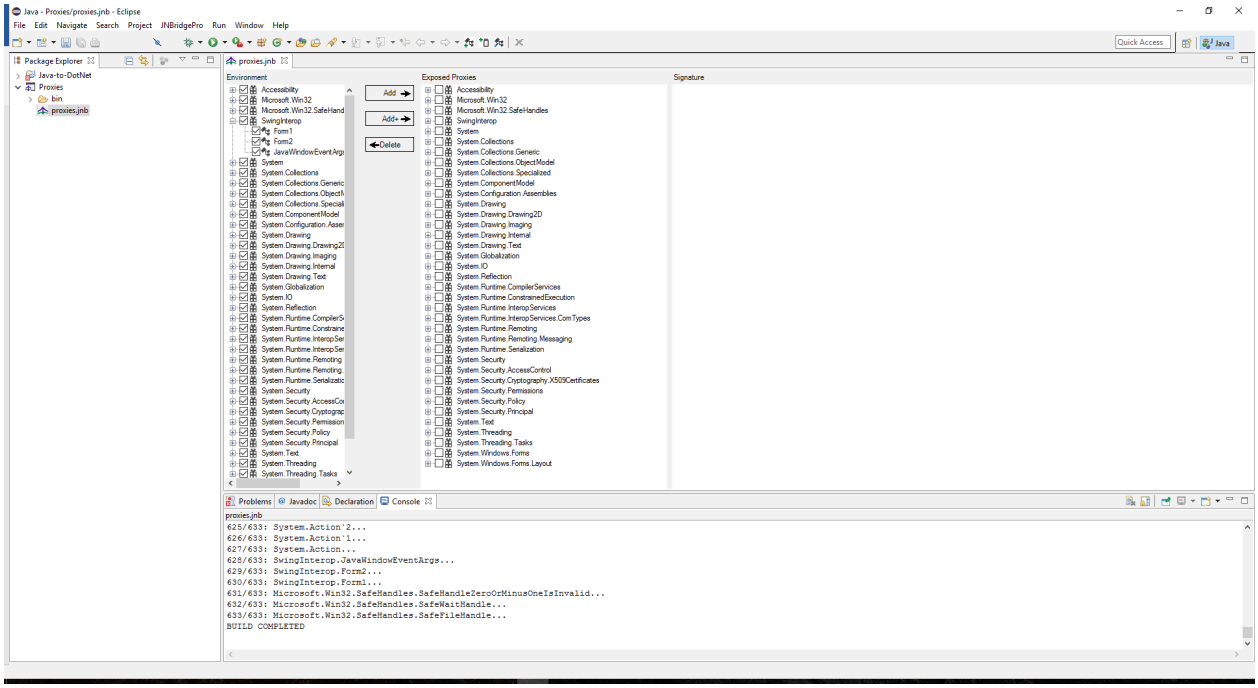

Figure 22. After adding classes to Exposed Proxies pane

Unlike the standalone proxy generation tool, we do not explicitly generate the proxies; they will be built automatically, as needed. In this case, for example, since Build Automatically was set, the build takes place whenever the contents of the Exposed Proxies pane changes, as it did here.

Using the proxies

To use the proxies in another project, right-click on that project's node in the Package Explorer and select **Build Path→Configure Build Path...**. Select the Libraries tab, then click on the "Add JARs…" button. A JAR Selection dialog box will be displayed. Navigate to the associated JNBridge project node, then into its bin directory and select the proxy jar file (Figure 24).

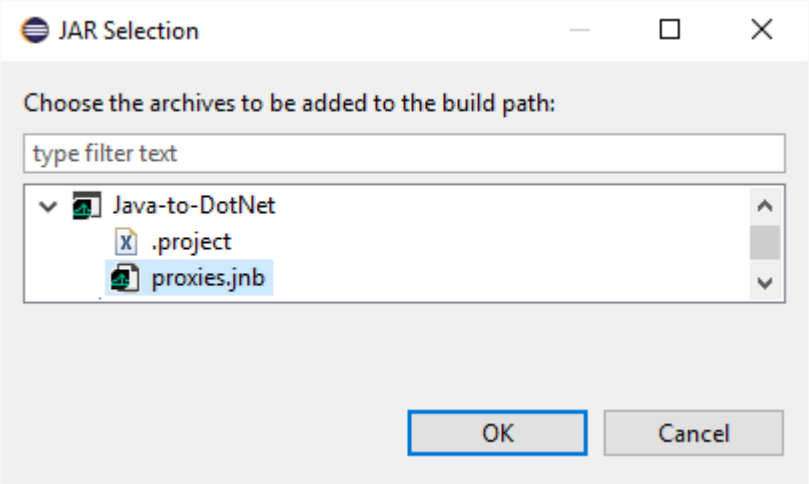

Figure 23. Adding a reference to the proxy jar file

One must also add the files jnbcore.jar and bcel-6.10.0.jar from the JNBridgePro installation to the Java Build Path. Then, when one performs a build, the referencing project will use the proxy jar file generated by the JNBridge project. If the JNBridge project is out of date or has not yet been built, it will automatically be build or rebuilt before being used.

Information concerning the generation of the proxies is displayed in Eclipse's Console window. If the build was unsuccessful, information describing the errors will be found there.