NuGet Lab instructions

The purpose of this lab is to familiarize yourself with the different NuGet tools available and to practice creating your own NuGet package.

Prerequisites

You must have:

- Visual Studio 2013 (or Visual Studio 2012) installed.
- The latest version of NuGet Package Manager installed in Visual Studio.
  (Visual Studio: Tools -> Extensions and Updates…)
- NuGet Command line tool. (http://nuget.org/nuget.exe)

Lab 1: Create and use simple NuGet package

In this lab we will create a simple NuGet package

Step 1: Create a class library project in Visual Studio

A. Create a new project in Visual Studio. Select 'Class Library' and call the project 'Common'
B. Create a simple class in the project. Call it 'Base'
C. Edit the AssemblyInfo.cs file. Set values on AssemblyDesciption and AssemblyCompany
D. Build the project.

Step 2: Create a NuGet package

A. Create a NuGet spec by running 'NuGet.exe spec' in the folder of your common project. Run 'NuGet.exe help' for more details.
B. Inspect the generated Common.nuspec file. Note that all values specified between $-signs will be extracted from assembly info.
C. Run 'NuGet.exe pack' to create the package.

A NuGet package has the file extension .nupkg but it really is a zip-file. Rename the package file with a .zip extension and look inside it. The .nuspec file that was used to create the package is included in the package. Open the file and note that the values specified with $-signs have been filled. Also look in the 'lib' folder of the package. Your compiled assembly is included in the package.

Step 3: Publish the package locally

NuGet can read packages from web services and directly from the file system. Here we create a local NuGet source.

A. Create a folder on the root of your hard drive and name it 'LocalNuGetSource' (C:\LocalNuGetSource)
B. Copy the package file to the folder

Step 4: Use the package

A. In Visual Studio, select Tools -> Library Package Manager -> Package Manager Settings. The options dialog appears. Create a new source with name = LocalSource and source = C:\LocalNuGetsource
B. Create a new solution/project with a Windows Console Application. Call it 'ClientOne'

C. Right-click the 'ClientOne' project and select 'Manage NuGet Packages'. In the dialog that appears select 'online/LocalSource'. You should see your 'Common' package
D. Select the package and click install then close the dialog.
E. You can now use the class 'Base' in you console application.

The following happened to your project to make this work:

1. The package was downloaded and unpacked to <solution>\packages.
2. A reference to the Common assembly was added (Path: ..\packages\Common.1.0.0.0\lib\net451\Common.dll)
3. A packages.config file was added to your project. This file tells NuGet what packages you have installed in your project.
4. A repositories.config file was added to <solution>\packages folder. This file tells NuGet what projects in your solution has any packages installed.

Step 5: Automatic restore during build

If any packages are missing, NuGet can automatically download and install them when you build your project. To enable this feature you do the following:

A. Right-click you solution (in Solution Explorer) and select 'Enable NuGet Package Restore'. NuGet will add a .nuget folder and a few files to your solution.
B. Test this by deleting the Common.1.0.0.0 folder from <solution>\packages and then rebuild your application

Note: When you are using automatic restore, the following files should be added to your source control:

- The .nuget folder and all files in that folder
- The repositories.config file in the packages folder.
- The packages.config file in each project.

Everything else is downloaded when you build. This means that (in our case, Common.1.0.0.0) any unpacked packages should **NOT** be added to source control.

Lab 2: NuGet package with dependencies

In this lab we will build a NuGet package with dependencies to other packages.

Step 1: Create a class library project in Visual Studio

A. Create a new project in Visual Studio. Select 'Class Library' and call the project 'A'
B. Install the 'Common' package in the project
C. Create a simple class, call it SubA and make it a sub class of Base.
D. Edit the AssemblyInfo.cs file. Set values on AssemblyDesciption and AssemblyCompany
E. Build the project.

Step 2: Create a NuGet package

Anyone who want to use the A package must also install the Common package as we have a direct dependency to it. This depencency is expressed in the NuGet spec.

A. Create a NuGet spec using NuGet.exe
B. Create the package using NuGet.exe.

C. Open the package and inspect the A.nuspec file inside. Note that a <dependencies> node was added automatically
D. Copy the created package to your local NuGet source

Step 3: Use the package

A. Create a new solution/project with a Windows Console Application. Call it ClientTwo
B. Install the A package.
C. Verify that the Common package was installed too. (Check 'references' and the packages folder)
D. Use the package visualizer (Tools -> Library Package Manager -> Package Vizualizer) to view all dependencies in your solution.

Lab 3: Updating a package

Version numbers

Version numbers on NuGet packages should follow the SemVer spec (http://semver.org/). A version should be three numbers, major.minor.patch followed by an optional pre-release label, like 1.2.3-beta1 or 2.0.0. (You can actually use any numbering scheme you want, but the SemVer spec is HIGHLY recommended.)

Step 1: Create a beta version of A

A. Open the 'A' solution and add a property to the SubA class
   int Foo { get; set; }
B. Build the project
C. Create a new package with version 2.0.0-beta1 using the NuGet command line tool:
   NuGet.exe pack -version 2.0.0-beta1
D. Copy the package to your local NuGet source

Step 2: Use the beta package

A. Open the 'ClientTwo' solution.
B. Right-click the solution in Solution Explorer and select 'Manage NuGet packages for solution…'
C. In the dialog, go to 'Updates' and select your local source. You should now see the beta version of A. If not, make sure 'Include Prerelease' is selected in the combo box above the list.
D. Click the 'Update' button. The beta version is installed in your solution and you can use the Foo property of the SubA class.

Step 3: Create a release version of A

A. Create a new package of A using the NuGet command line tool:
   NuGet.exe pack -version 2.0.0
B. Copy the package to your local NuGet source

Step 4: Update to the release version of A

Here we will update to the release version of A, but we will use the Package Manager Console instead of the Package dialog. The console is a Powershell window with 'intellisense' on the tab-key.

A. Open the 'ClientTwo' solution
B. Select Tools -> Library Package Manager -> Package Manager Console
C. Type *get-package* at the command prompt. This will list installed packages.

D. Make sure ‘Local source’ is selected in the Package Source combo in the top of the window.
E. Type *update-package*. This will update all packages in your solution to the latest stable versions. (If you want to include pre-release package add the ‘-IncludePrerelease’ option)
F. Note the output of the command. Version 2.0.0 was installed and version 2.0.0-beta1 was uninstalled.

Conclusion

NuGet is a powerful tool with great integration in Visual Studio. More information can be found here:

http://docs.nuget.org/