nik codes

23 Oct 2013
12 Comments

Packaging Source Code With NuGet

.NET developers depend on binaries **a lot**. Assemblies and executables are our typical unit of deployment and consumption and NuGet itself is basically a sophisticated binary distribution system.

I don't think there is anything *necessarily* wrong with that. Assemblies offer a lot of power and have plenty of benefits, but a few times over the last year or so I've wondered if our assembly usage follows the law of the instrument (https://web.archive.org/web/20150220161402/http://en.wikipedia.org/wiki/Law_of_the_instrument).

"I suppose it is tempting, if the only tool you have is a hammer, to treat everything as if it were a nail. - Abraham Maslow"

Plenty of other development communities very happily exist in a world without assemblies or binaries. The Ruby gems (https://web.archive.org/web/20150220161402/http://rubygems.org/) and npm (https://web.archive.org/web/20150220161402/https://npmjs.org/) repositories are absolutely filled to the brim with packages that contain no binary files whatsoever. What do they contain? **Source code.**

Recently I've been wanting to create a NuGet package that would contain common AutoFixture (https://web.archive.org/web/20150220161402/https://github.com/autofixture) configuration for intrinsic ASP.NET types that I find myself repeatedly using. Unfortunately, every other user of the package would want

to slightly tweak the configuration for their project, so I figured I'd simply ship the package with only the source code (no pre-compiled assembly) so that they could easily do that.

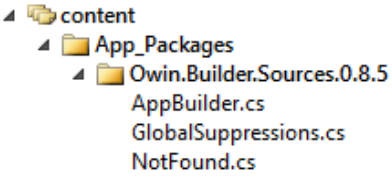
When I told Louis DeJardin (https://web.archive.org/web/20150220161402/http://whereslou.com/) about this idea earlier in the year at MonkeySpace (https://web.archive.org/web/20150220161402/http://monkeyspace.org/), his eyes lit up and he got excited. It turns out he's been thinking about source only packages quite a bit and has even released several (https://web.archive.org/web/20150220161402/http://www.nuget.org/packages/Owin.Types.Sources/) OWIN (https://web.archive.org/web/20150220161402/http://www.nuget.org/packages/Owin.Loader.Sources/) packages (https://web.archive.org/web/20150220161402/http://www.nuget.org/packages/Owin.Extensions.Sources/) as source only.

Louis told me about a few best practices and conventions that he has found useful when creating source only packages. I've noticed that only Damian Hickey (https://web.archive.org/web/20150220161402/http://dhickey.ie/) has picked up on these, so with Louis' permission, I've decided to publish them here for greater exposure:

NuGet Source-Only Package Conventions and Practices

1. Package ID's should end in `.Sources`, which aligns well with and builds upon the `.Sample` suffix recommended in the NuGet Package Conventions documentation (https://web.archive.org/web/20150220161402/http://docs.nuget.org/docs/creating-packages/package-conventions).
2. In the `.nupkg` itself, source code files should be placed in a directory with the path `content/App_Packages/{package-id}.{package-version}`. Similar to how the `.Sources` suffix builds upon the prior art of `.Samples`, the `App_Packages` directory follows the `App_Start` folder nomenclature used by David Ebbo's (https://web.archive.org/web/20150220161402/http://blog.davidebbo.com/) popular WebActivator (https://web.archive.org/web/20150220161402/https://github.com/davidebbo/WebActivator) package.

   Here's an example of a package following this convention, as seen in NuGet Package Explorer (https://web.archive.org/web/20150220161402/http://npe.codeplex.com/):

   (https://web.archive.org/web/20150220161402/http://nikmd23.files.wordpress.com/2013/10/path.png)
   This convention also allows for a very convenient upgrade path which I'll cover later on.

3. Source-Only packages can depend on any other package, including other Source-Only packages.
4. Source-Only packages may leverage NuGet's Source Code Transformations (https://web.archive.org/web/20150220161402/http://docs.nuget.org/docs/creating-packages/configuration-file-and-source-code-transformations) (`*.pp` files) to inject project properties into the source. This is most often seen with the use of the **`$rootNamespace$`** property, but any project property (https://web.archive.org/web/20150220161402/http://msdn.microsoft.com/en-us/library/vslangproj.projectproperties_properties%28VS.80%29.aspx) can be used.
5. In some situations, it may be useful to also ship a standard "binary" package in addition to the Source-Only package.
6. Types in Source-Only packages should be marked as internal by default in order to not pollute the target code's public API.

7. Consider using conditional compilation symbols or partial classes in Source-Only packages to provide flexibility and the ability to customize the source to users.

   Examples of this technique include allowing for multiple platform targeting options and changing the accessibility of types to public when desired. SimpleJson has a few good examples of this in their source code (https://web.archive.org/web/20150220161402/https://github.com/facebook-csharp-sdk/simple-json/blob/master/src/SimpleJson/SimpleJson.cs).

When to Create Source-Only Packages

There are plenty of situations where a Source-Only package does not make sense. Here's a few things to consider:

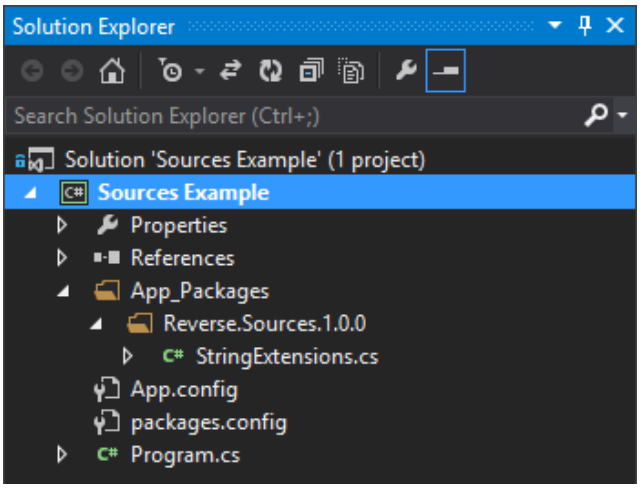
- **DO** consider creating a Source-Only package for "utility" libraries that feature heavy usage of static and/or extension methods. Examples of these types of utility libraries include unit test assertion libraries (see note below!) and the popular DataAnnotationsExtensions (https://web.archive.org/web/20150220161402/http://www.nuget.org/packages/DataAnnotationsExtensions/) package.
- **DO** consider creating a Source-Only package for small single purpose libraries. SimpleJson (https://web.archive.org/web/20150220161402/http://www.nuget.org/packages/SimpleJson/) is already doing this (though not following these conventions) but you can image any code appropriate for a blog post or Gist would fit the definition well. (*Aside: a Gist to Source-Only NuGet Package service would be pretty useful!*)
- **DO** consider creating a Source-Only package for common configuration and setup code or any code which will *require* tweaking by the user.
- **DO NOT** consider creating a Source-Only package as a means to simply make step-debugging easier. Instead leverage a symbols package (https://web.archive.org/web/20150220161402/http://docs.nuget.org/docs/creating-packages/creating-and-publishing-a-symbol-package).

Source-Only Package Update Process

One of the nice things about assemblies is that the process of versioning them is well understood. How do we version and update source-only NuGet packages? Luckily, NuGet's native handling of content works in our favor. Let's explore with an example:

I've created a source-only package called `Reverse.Sources` (https://web.archive.org/web/20150220161402/https://www.nuget.org/packages/Reverse.Sources/) that contains an extension method for reversing a string. Let's install it:

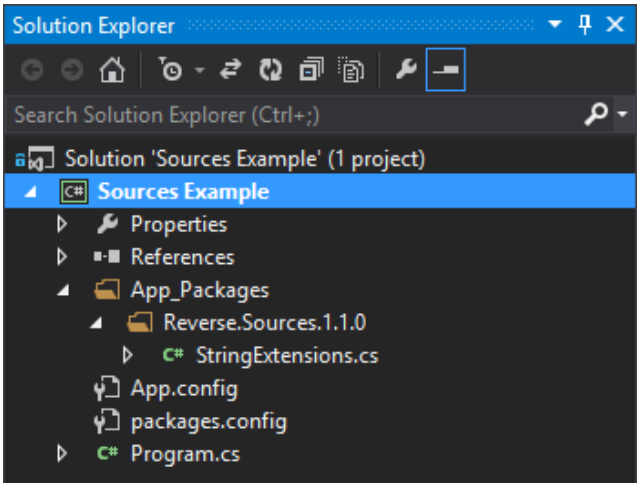
```
Install-Package Reverse.Sources -Version 1.0.0
```



Great, now our project contains that `StringExtensions.cs` file with an implementation to reverse strings. It compiles right along with our application and is relatively out of our way.

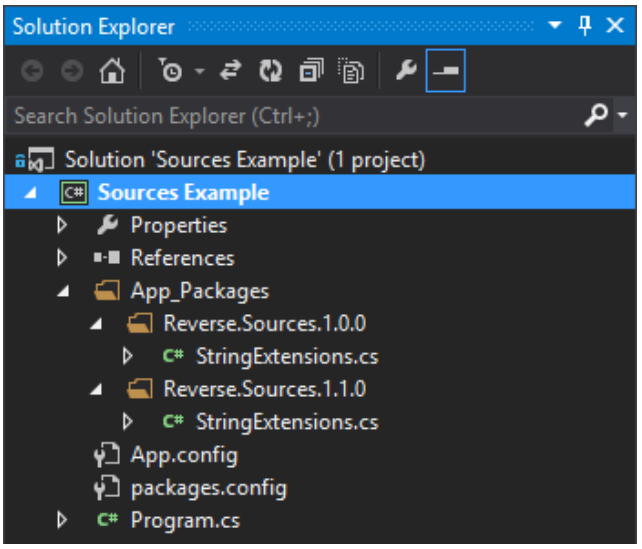
Unfortunately, version 1.0.0 of my package had a bug and blows up if a null string is passed into the Reverse method. I've added a simple guard clause to fix the problem and released it in version 1.1.0. Let's update:

```
Update-Package Reverse.Sources
```



Notice Solution Explorer looks nearly identical – Reverse.Sources.1.0.0 was simply replaced, along with all of it's files, by Reverse.Sources.1.1.0. I've updated my project without any troubles and I have that nice bug fix now.

But what if we had made changes to `StringExtensions.cs`? NuGet would have simply left behind the files you've edited.



We'd know that there was a problem too because the compiler would complain with a *"The namespace 'Sources_Example' already contains a definition for 'StringExtensions'"* error.

To fix that error we can use a text diff/merge tool to move the changes over and delete the old 1.0.0 folder.

To me this is a pretty clean upgrade process. Sure we could sometimes get into a situation where we have to deal with merging, but we also get the benefits of working directly with the code.

Are Source-Only Packages A Bad Idea?

Perhaps yes, perhaps no. I don't think they are that crazy of an idea though. Large numbers of developers outside the .NET ecosystem already work almost exclusively with source only packages. Further, I'd propose that you do as well if you've ever included jQuery, Modernizr or any other JavaScript NuGet package in your project.

I for one probably wouldn't want all packages to be source-only (we certainly won't be shipping Glimpse like this!), but there are situations where I think it could be extremely useful – particularly in scenarios involving extensibility/tweaking and reducing dependency overhead when I might have reached for ILMerge otherwise.

I'm hoping that this post can start a bit of a conversation about the idea of source only packages and increase the community's comfort with them. I'm interested to hear the thoughts of the NuGet team and will be submitting these conventions to the NuGet documentation shortly.

Note: It looks like this is on Brad Wilson's mind too!

> Brad Wilson (on indefinite hiatus) @bradwilson
> Part of me wonders if @xunit v2 should ship Assert as NuGet-based source instead of binary, so you can easily extend it (or ignore it).
> 1
> 1:59 PM - 1 Sep 2013
> Twitter Ads info and privacy
>
> See Brad Wilson (on indefinite hiatus)'s other Tweets

Updated Oct 25th with feedback from Phil Haack (https://web.archive.org/web/20150220161402/http://haacked.com/), Louis DeJardin, Prabir Shrestha (https://web.archive.org/web/20150220161402/http://blog.prabir.me/) and Andrew Nurse

(https://web.archive.org/web/20150220161402/http://flavors.me/anurse). Thanks for the feedback guys!

About these ads (https://web.archive.org/web/20150220161402/http://wordpress.com/about-these-ads/)
Posted in NuGet
← Older Entry (https://web.archive.org/web/20150220161402/http://nikcodes.com/2013/10/09/enjoy-the-perks-of-open-source-development/)
Newer Entry → (https://web.archive.org/web/20150220161402/http://nikcodes.com/2014/03/04/flushing-in-asp-net-mvc/)

12 thoughts on “Packaging Source Code With NuGet”

1.  JP Toto (@jptoto) on October 25, 2013 at 9:38 am said:
Great thoughts, Nik! I like the conventions that were laid out too. I'll have to fiddle with this a little to see how it shakes out for some libs I work on.

Reply ↓ (/web/20150220161402/http://nikcodes.com/2013/10/23/packaging-source-code-with-nuget/?replytocom=946#respond)

2.  nikmd23 on October 25, 2013 at 11:21 am said:
Thanks JP. Be sure to let me know how things work out in your fiddling.

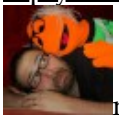
Reply ↓ (/web/20150220161402/http://nikcodes.com/2013/10/23/packaging-source-code-with-nuget/?replytocom=947#respond)

3.  Haacked on October 25, 2013 at 11:30 am said:
One thing you didn't mention is that if you create a source package, you should make the types internal by default so as not to pollute the target code's public API.

Also, use conditional compilation symbols to provide some flexibility such as multiple platform targeting and changing types to public.

For a great example of a source package, check out https://github.com/facebook-csharp-sdk/simple-json (https://web.archive.org/web/20150220161402/https://github.com/facebook-csharp-sdk/simple-json)

Reply ↓ (/web/20150220161402/http://nikcodes.com/2013/10/23/packaging-source-code-with-nuget/?replytocom=948#respond)

4.  nikmd23 on October 25, 2013 at 1:22 pm said:
Phil,

Thanks so much for your feedback. I've updated the post to incorporate it now!

Reply ↓ (/web/20150220161402/http://nikcodes.com/2013/10/23/packaging-source-code-with-nuget/?replytocom=949#respond)

5.  David De Sloovere (@DavidDeSloovere) on October 25, 2013 at 2:57 pm said:
I was just thinking about the $RootNamespace$ property that gets replaced upon installation of the package.

   Having this in your code obviously wouldn't allow you to actually build the files you plan to put in the package. So you might have something that has a syntax error without you really knowing it.

   But if you used a real namespace and you were able to change that namespace into $RootNamespace$ right before packaging, that would be solved. Some msbuild regex mojo or maybe powershell?

   Would also be useful to packages up a whole MVC area, with controllers and also views. With MVC5's attribute routing you don't even need the AreaRegistration anymore!

   Reply ↓ (/web/20150220161402/http://nikcodes.com/2013/10/23/packaging-source-code-with-nuget/?replytocom=950#respond)

   -  nikmd23 on October 25, 2013 at 3:02 pm said:
   That's a great idea David. Some sort of token replacement process that would allow you to freely develop/compile but still ship with the placeholders would be nice.

     To be clear through, I don't actually think using $RootNamespace$ in a Source-Only package is the right thing to do in most situations. I've called out that example because it is the most used of the source transforms that I've seen and wanted to provide some context on the idea for the uninitiated.

     Reply ↓ (/web/20150220161402/http://nikcodes.com/2013/10/23/packaging-source-code-with-nuget/?replytocom=951#respond)

     -  David De Sloovere (@DavidDeSloovere) on October 25, 2013 at 3:37 pm said:
     Would you rather not put a namespace around your class then or have a fixed one? I'm lean heavily toward the convention of having your namespace follow the folder structure, but I guess that would be a problem anyway when applying the version naming like "Reverse.Sources.1.1.0".

       Personally I haven't used any other property transform either, so that the first and only one I can think of. But your are right, shouldn't be limited to this one or even be this any way.

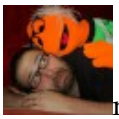
       And here's a blog post by Daniel "Kzu" about replacing content in files in MSBuild:

       http://blogs.clariusconsulting.net/kzu/how-to-perform-regular-expression-based-replacements-on-files-with-msbuild/ (https://web.archive.org/web/20150220161402/http://blogs.clariusconsulting.net/kzu/how-to-perform-regular-expression-based-replacements-on-files-with-msbuild/)

6.  hullah on October 25, 2013 at 7:19 pm said:
I like this idea when used appropriately. I've seen other Nuget source only packages that was one source file but contained many, many nested classes. That was kind of out of control.

   I also like the idea of being able to extend a source only package with partial classes. That way you could add additional functionality to the supplied classes without modifying the original source to allow for smoother upgrades.

   Reply ↓ (/web/20150220161402/http://nikcodes.com/2013/10/23/packaging-source-code-with-nuget/?replytocom=954#respond)

7.  nikmd23 on October 28, 2013 at 10:17 am said:
@David – I'm also a fan of matching directory structures and namespaces, but we don't do this for libraries right now anyways.

I do think there is times to have the package code's namespace match the user's namespace (by leveraging $rootNamespace$), but I think that most of the time having a package specific namesspace is probably better. Fixed namespaces also allow the user to freely switch between the Source-Only and binary package.

@hullah – yes, this technique could be abused for sure. Let the author of the package know that you'd appreciate the code to be separated across files. I bet you'd be surprised by their willingness to comply, or provide an excellent reason for why one source file is preferable in that situation.

Reply ↓ (/web/20150220161402/http://nikcodes.com/2013/10/23/packaging-source-code-with-nuget/?replytocom=960#respond)

8.  pascalberger on May 15, 2014 at 10:00 am said:
Just stumbled across your post while trying to create some source only packages. In my use case I don't want to provide a set of default files, which are later changed in the project, but some extensions and static helper classes which should not change in the project.
The NuGet workflow copies files contained in the content folder to the project, and the files need to be added to source control, since package restore won't copy the files. This may lead that other developers try to change these files in a project, instead of changing the NuGet package and releasing a new version of it.

Since I see that there are use cases for this workflow I unfortunately haven't found a good way how to deal with this in my case. Adding the files as links to the package directory might be a solution (instead copying them), but unfortunately I wast't able yet to get this running.

Reply ↓ (/web/20150220161402/http://nikcodes.com/2013/10/23/packaging-source-code-with-nuget/?replytocom=1809#respond)

-  nikmd23 on May 15, 2014 at 10:18 am said:
What is your reasoning for wanting to release as a source only package?

Sounds like a standard binary package would work well for your requirements.

Thoughts?

Reply ↓ (/web/20150220161402/http://nikcodes.com/2013/10/23/packaging-source-code-with-nuget/?replytocom=1810#respond)

-  pascalberger on May 15, 2014 at 11:05 am said:
Normally yes :-)

But I need to for SharePoint projects, which is a little bit a different situation than with standard ASP.NET or desktop applications. In this case binary packages would result in some problem which source only package would solve:
– SharePoint has it's own packaging system (WSP files). After adding a reference to a SharePoint project, the package needs also manually be added to the SharePoint project, so that it is deployed to the GAC on the SharePoint server. Unfortunately Visual Studio cannot automatically pack all required assembly into the WSP files.
– With binary packages there might be version conflicts. Consider solution A which needs core library

in version 1. Later someone develops solution B against core library version 2. Depending on the customer this might be quite a big problem, if they have very strong guidelines what can be deployed to their farm.

Source only packages would solve both problems quite elegantly. But the cost of needing to add the files to source control in each project is also not worth it.