How can I diagnose missing dependencies (or other loader failures) in dnx?

Asked 9 years, 9 months ago   Modified 2 months ago   Viewed 53k times

133

I'm trying to run a modified version of the HelloWeb sample for ASP.NET vNext on DNX using Kestrel. I understand that this is *very* much on the bleeding edge, but I would hope that the ASP.NET team would at least keep the simplest possible web app working :)

Environment:

- Linux (Ubuntu, pretty much)
- Mono 3.12.1
- DNX 1.0.0-beta4-11257 (I have 11249 available too)

"Web app" code, in `Startup.cs`:

```
using Microsoft.AspNet.Builder;
public class Startup
{
    public void Configure(IApplicationBuilder app)
    {
        app.UseWelcomePage();
    }
}
```

Project config, in `project.json`:

```
{
  "dependencies": {
    "Kestrel": "1.0.0-beta4",
    "Microsoft.AspNet.Diagnostics": "1.0.0-beta4",
    "Microsoft.AspNet.Hosting": "1.0.0-beta4",
    "Microsoft.AspNet.Server.WebListener": "1.0.0-beta4",
    "Microsoft.AspNet.StaticFiles": "1.0.0-beta4",
    "Microsoft.Framework.Runtime": "1.0.0-beta4",
    "Microsoft.Framework.Runtime.Common": "1.0.0-beta4",
    "Microsoft.Framework.Runtime.Loader": "1.0.0-beta4",
    "Microsoft.Framework.Runtime.Interfaces": "1.0.0-beta4",
  },
  "commands": {
    "kestrel": "Microsoft.AspNet.Hosting --server Kestrel --server.urls
http://localhost:5004"
  },
  "frameworks": {
    "dnx451": {}
    }
  }
```

`kpm restore` appears to work fine.

When I try to run, however, I get an exception suggesting that `Microsoft.Framework.Runtime.IApplicationEnvironment` can't be found. Command line and error (somewhat reformatted)

```
.../HelloWeb$ dnx . kestrel
System.IO.FileNotFoundException: Could not load file or assembly
'Microsoft.Framework.Runtime.IApplicationEnvironment,
  Version=0.0.0.0, Culture=neutral, PublicKeyToken=null'
or one of its dependencies.
File name: 'Microsoft.Framework.Runtime.IApplicationEnvironment,
  Version=0.0.0.0, Culture=neutral, PublicKeyToken=null'
  at (wrapper managed-to-native) System.Reflection.MonoMethod:InternalInvoke
    (System.Reflection.MonoMethod,object,object[],System.Exception&)
  at System.Reflection.MonoMethod.Invoke
    (System.Object obj, BindingFlags invokeAttr, System.Reflection.Binder
binder,
     System.Object[] parameters, System.Globalization.CultureInfo culture)
    [0x00000] in <filename unknown>:0
```

While obviously, my most pressing need is to fix this, I'd also appreciate advice on how to move to diagnose what's going wrong so I can fix similar issues myself in the future. (That's also likely to make this question more useful to others, too.)

I've found `Microsoft.Framework.Runtime.IApplicationEnvironment` in the `Microsoft.Framework.Runtime.Interfaces` assembly source, and that doesn't appear to have changed recently. It's not clear why the exception shows the name as if it's a whole assembly in itself, rather than just an interface within another assembly. ~~I'm guessing this *may* be due to assembly neutral interfaces, but it's not clear from the error.~~ (`[AssemblyNeutral]` is dead, so that's not it...)

c# asp.net-core dnx

Share

Improve this question

Follow

edited Mar 1, 2019 at 8:21

ArunPratap
5,010 ● 7 ● 28 ● 45

asked Mar 12, 2015 at 11:00

Jon Skeet
1.5m ● 889 ● 9.3k ● 9.3k

out of curiosity, did you mean to link to github.com/aspnet/Home/wiki/Assembly-Neutral-Interfaces for your assembly neutral interfaces link or somewhere else? As it is currently broken – cgijbels Mar 12, 2015 at 11:12

@cgijbels: Thanks - I actually meant to link to davidfowl.com/assembly-neutral-interfaces but your link it probably better... – Jon Skeet Mar 12, 2015 at 11:19

Assembly loading failures like you show in your question are in my experience usually due to your DNX and/or your packages being from different build versions. Verifying that the packages downloaded by kpm restore are all the same version, and that this version matches your DNX should be your first step towards debugging these. I'm not sure what the dependency resolving mechanism does when it can't find the 1.0.0-beta4 package version. No package like that exists, so it should have failed. Sounds a bit like a bug to me? – AndersNS Mar 12, 2015 at 14:19

@AndersNS: No, the packages *do* exist, in the nightly build repo. myget.org/gallery/aspnetvnext – Jon Skeet Mar 12, 2015 at 14:20

Packages versioned with "1.0.0-beta4-*buildnumber*" exists. I can't see "1.0.0-beta4" up there. Packages without a build number won't be released until beta4 is deemed "stable-ish" and merged into master, if they do it like they did for beta3, beta 2 and beta1. – AndersNS Mar 12, 2015 at 14:32

6 Answers

Sorted by: Highest score (default)

143

+200

For your specific problem, it looks like you have a mismatch in your resolved dependencies. When things like this happen it's likely because you're running your application on an incompatible dnx. We're still making very big breaking changes so if you ever see method missing of type missing, chances are you ended up running `betaX` packages and `betaY` dnx or vice versa.

Even more specifically, Assembly Neutral Interfaces were removed in beta4 but it looks like the application you are running is still using them.

We have plans to make it so that packages can mark the minimum dnx that they require to run to make the error message more clear. Also as time goes by, the breaking changes will die down.

In general though, I feel like it's time I wrote a guide on how to diagnose issues like this when using the dnx (since it's pretty different to existing .NET).

Dependencies you put into `project.json` are top level only. Versions are also **always minimums** (it's just like a NuGet package). This means that when you specify `Foo 1.0.0-beta4` you're really specifying `Foo >= 1.0.0-beta4`. This means if you ask for `MVC 0.0.1` and the minimum versions on your configured feed is `MVC 3.0.0`, you'll get that one. We also *NEVER* float your version unless you specify it. If you ask for 1.0.0 and it exists, you will get 1.0.0 even if newer versions exist. Specifying empty versions is *ALWAYS* bad and will be disallowed in later builds.

There's a new feature we're introducing to NuGet called floating versions. Today it only works on the prerelease tag, but in the next version it'll work on more parts of the version. This is similar to npm and gem syntax for specifying version ranges in the package specification file.

`1.0.0-*` - Means give me the HIGHEST version matching the prefix (according to semantic versioning rules) OR if there is no version matching that prefix, use normal behavior and get me the LOWEST version >= the specified version.

When you run restore in the latest builds, it will write out a file called `project.lock.json`. This file will have the transitive closure of dependencies for all target frameworks defined in `project.json`.

When something like this fails you can do the following:

Take a look at the resolved dependencies using `kpm list`. This will show you the resolved versions of packages referenced by your project and what dependency pulled it in. e.g. if A -> B, it'll show:

```
A
  -> B
B
 ->
```

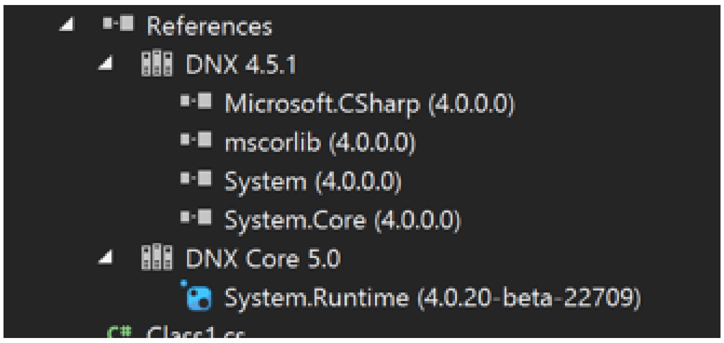
Actual KPM list output:

Listing dependencies for ClassLibrary39 (C:\Users\davifowl\Documents\Visual Studio 14\Projects\ClassLibrary39\src\ClassLibrary39\project.json)

```
[Target framework DNX,Version=v4.5.1 (dnx451)]

 framework/Microsoft.CSharp 4.0.0.0
    -> ClassLibrary39 1.0.0
 framework/mscorlib 4.0.0.0
    -> ClassLibrary39 1.0.0
 framework/System 4.0.0.0
    -> ClassLibrary39 1.0.0
 framework/System.Core 4.0.0.0
    -> ClassLibrary39 1.0.0
*Newtonsoft.Json 6.0.1
    -> ClassLibrary39 1.0.0

[Target framework DNXCore,Version=v5.0 (dnxcore50)]

*Newtonsoft.Json 6.0.1
    -> ClassLibrary39 1.0.0
 System.Runtime 4.0.20-beta-22709
    -> ClassLibrary39 1.0.0
```

* means direct dependency.

If you have a working visual studio (which breaks with DNX right now), you can look at the references node. It has the same data represented visually:



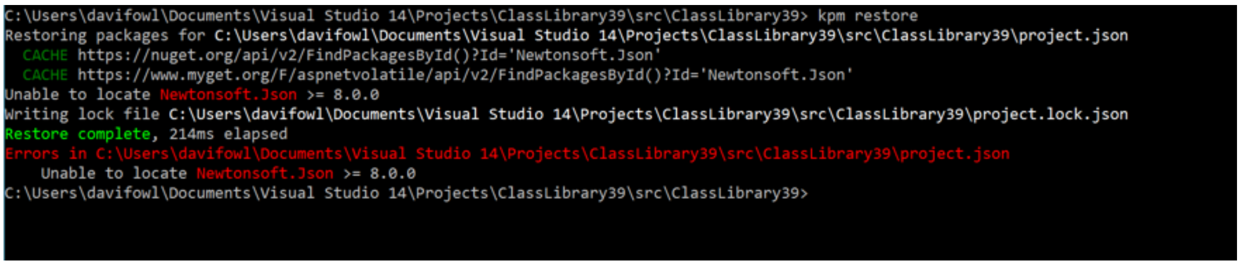
```
References
  DNX 4.5.1
    Microsoft.CSharp (4.0.0.0)
    mscorlib (4.0.0.0)
    System (4.0.0.0)
    System.Core (4.0.0.0)
  DNX Core 5.0
    System.Runtime (4.0.20-beta-22709)
```

Let's look at what a dependency failure looks like:

Here's the project.json

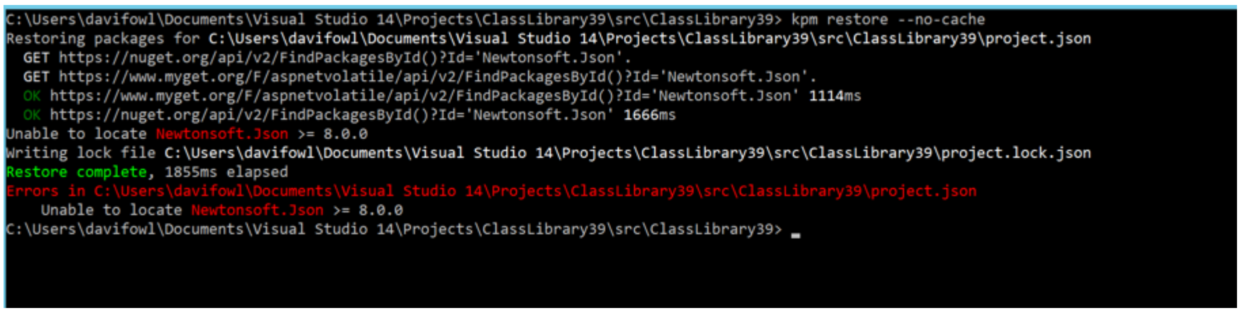
```
{
    "version": "1.0.0-*",
    "dependencies": {
        "Newtonsoft.Json": "8.0.0"
    },

    "frameworks" : {
        "dnx451" : {
            "dependencies": {
            }
        },
        "dnxcore50" : {
            "dependencies": {
                "System.Runtime": "4.0.20-beta-22709"
            }
        }
    }
}
```

`Newtonsoft.Json 8.0.0` doesn't exist. So running kpm restore shows the following:



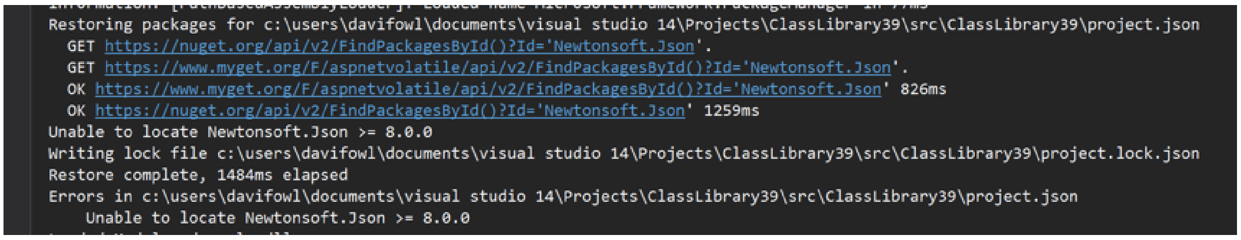
```
C:\Users\davifowl\Documents\Visual Studio 14\Projects\ClassLibrary39\src\ClassLibrary39> kpm restore
Restoring packages for C:\Users\davifowl\Documents\Visual Studio 14\Projects\ClassLibrary39\src\ClassLibrary39\project.json
  CACHE https://nuget.org/api/v2/FindPackagesById()?Id='Newtonsoft.Json'
  CACHE https://www.myget.org/F/aspnetvolatile/api/v2/FindPackagesById()?Id='Newtonsoft.Json'
Unable to locate Newtonsoft.Json >= 8.0.0
Writing lock file C:\Users\davifowl\Documents\Visual Studio 14\Projects\ClassLibrary39\src\ClassLibrary39\project.lock.json
Restore complete, 214ms elapsed
Errors in C:\Users\davifowl\Documents\Visual Studio 14\Projects\ClassLibrary39\src\ClassLibrary39\project.json
    Unable to locate Newtonsoft.Json >= 8.0.0
C:\Users\davifowl\Documents\Visual Studio 14\Projects\ClassLibrary39\src\ClassLibrary39>
```

When diagnosing when restore might have failed, look at the HTTP requests made, they tell you what configured package sources kpm looked in. Notice in the above image, there is a `CACHE` request. This is the built in caching based on the type of resource (nupkg or nuspec) and has a configurable TTL (look at `kpm restore --help`). If you want to force `kpm` to hit the remote NuGet sources, use the `--no-cache` flag:



```
C:\Users\davifowl\Documents\Visual Studio 14\Projects\ClassLibrary39\src\ClassLibrary39> kpm restore --no-cache
Restoring packages for C:\Users\davifowl\Documents\Visual Studio 14\Projects\ClassLibrary39\src\ClassLibrary39\project.json
  GET https://nuget.org/api/v2/FindPackagesById()?Id='Newtonsoft.Json'.
  GET https://www.myget.org/F/aspnetvolatile/api/v2/FindPackagesById()?Id='Newtonsoft.Json'.
  OK https://www.myget.org/F/aspnetvolatile/api/v2/FindPackagesById()?Id='Newtonsoft.Json' 1114ms
  OK https://nuget.org/api/v2/FindPackagesById()?Id='Newtonsoft.Json' 1666ms
Unable to locate Newtonsoft.Json >= 8.0.0
Writing lock file C:\Users\davifowl\Documents\Visual Studio 14\Projects\ClassLibrary39\src\ClassLibrary39\project.lock.json
Restore complete, 1855ms elapsed
Errors in C:\Users\davifowl\Documents\Visual Studio 14\Projects\ClassLibrary39\src\ClassLibrary39\project.json
    Unable to locate Newtonsoft.Json >= 8.0.0
C:\Users\davifowl\Documents\Visual Studio 14\Projects\ClassLibrary39\src\ClassLibrary39> _
```

These errors also show up in Visual Studio in the package manager log output window:



```
Restoring packages for c:\users\davifowl\documents\visual studio 14\Projects\ClassLibrary39\src\ClassLibrary39\project.json
  GET https://nuget.org/api/v2/FindPackagesById()?Id='Newtonsoft.Json'.
  GET https://www.myget.org/F/aspnetvolatile/api/v2/FindPackagesById()?Id='Newtonsoft.Json'.
  OK https://www.myget.org/F/aspnetvolatile/api/v2/FindPackagesById()?Id='Newtonsoft.Json' 826ms
  OK https://nuget.org/api/v2/FindPackagesById()?Id='Newtonsoft.Json' 1259ms
Unable to locate Newtonsoft.Json >= 8.0.0
Writing lock file c:\users\davifowl\documents\visual studio 14\Projects\ClassLibrary39\src\ClassLibrary39\project.lock.json
Restore complete, 1484ms elapsed
Errors in c:\users\davifowl\documents\visual studio 14\Projects\ClassLibrary39\src\ClassLibrary39\project.json
    Unable to locate Newtonsoft.Json >= 8.0.0
```

Side note!

Package Sources

I'll describe the way NuGet.config works right now (which will likely change in the future). By default you have a NuGet.config with the default NuGet.org source configured globally in `%appdata%\NuGet\NuGet.Config`. You can manage these global sources within visual studio or with the NuGet command line tool. You should always look at your effective sources (the ones listed in the kpm output) when trying to diagnose failures.

Read more about NuGet.config here

Back to reality:

When dependencies are unresolved, running the application will give you this:

```
> dnx . run
System.InvalidOperationException: Failed to resolve the following dependencies
for target framework 'DNX,Version=v4.5.1':
   Newtonsoft.Json 8.0.0

Searched Locations:
  C:\Users\davifowl\Documents\Visual Studio 14\Projects\ClassLibrary39\src\
{name}\project.json
  C:\Users\davifowl\Documents\Visual Studio 14\Projects\ClassLibrary39\test\
{name}\project.json
  C:\Users\davifowl\.dnx\packages\{name}\{version}\{name}.nuspec
  C:\Program Files (x86)\Reference
Assemblies\Microsoft\Framework\.NETFramework\v4.5.1\{name}.dll
  C:\Program Files (x86)\Reference
Assemblies\Microsoft\Framework\.NETFramework\v4.5.1\Facades\{name}.dll
  C:\WINDOWS\Microsoft.NET\assembly\GAC_32\{name}\{version}\{name}.dll
```

```
    C:\WINDOWS\Microsoft.NET\assembly\GAC_64\{name}\{version}\{name}.dll
    C:\WINDOWS\Microsoft.NET\assembly\GAC_MSIL\{name}\{version}\{name}.dll

Try running 'kpm restore'.

   at Microsoft.Framework.Runtime.DefaultHost.GetEntryPoint(String
applicationName)
   at Microsoft.Framework.ApplicationHost.Program.ExecuteMain(DefaultHost host,
String applicationName, String[] args)
   at Microsoft.Framework.ApplicationHost.Program.Main(String[] args)
```

The runtime basically tries to validate that the entire dependency graph is resolved before attempting to run. If it suggests running `kpm restore` it's because it can't find the dependencies listed.

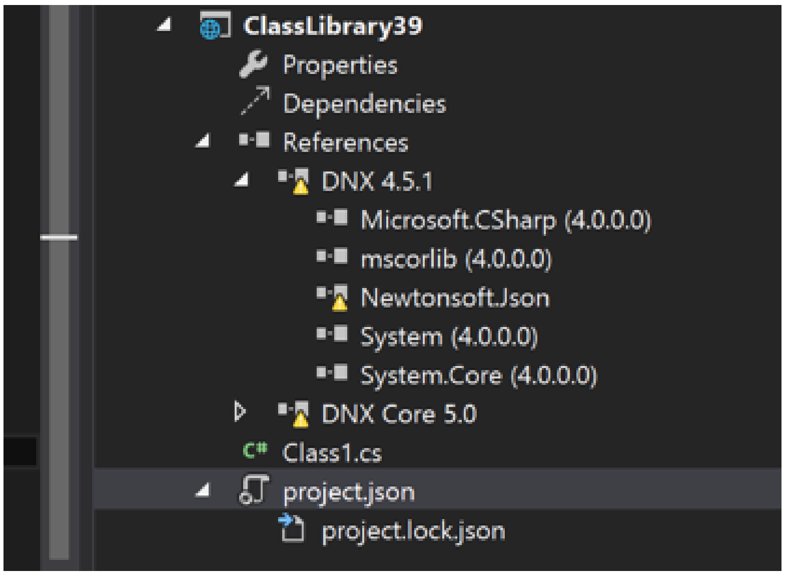
Another reason why you might get this error is if you're running the wrong dnx flavor. If your application only specifies dnx451 and you try to run the CoreCLR dnx, you might see a similar problem. Pay close attention to the target framework in the error message:

For running:

```
dnx4x - runs on dnx-clr-{etc}
dnxcore50 - runs on dnx-coreclr-{etc}
```

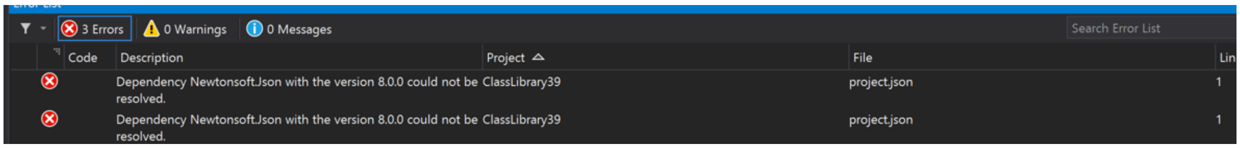
When you're trying to run, you should remember that mental mapping from clr to target framework defined in your `project.json`.

This also shows up in Visual Studio under the references node:



The nodes marked as yellow are unresolved.

These also show up in the error list:



	Code	Description	Project	File	Lin
⊗		Dependency Newtonsoft.Json with the version 8.0.0 could not be resolved.	ClassLibrary39	project.json	1
⊗		Dependency Newtonsoft.Json with the version 8.0.0 could not be resolved.	ClassLibrary39	project.json	1

Building

These errors also show up when building. When building from the command line, the output is very verbose and can be extremely useful when diagnosing problems:

```
> kpm build

Building ClassLibrary39 for DNX,Version=v4.5.1
  Using Project dependency ClassLibrary39 1.0.0
    Source: C:\Users\davifowl\Documents\Visual Studio
14\Projects\ClassLibrary39\src\ClassLibrary39\project.json

  Using Assembly dependency framework/mscorlib 4.0.0.0
    Source: C:\Program Files (x86)\Reference
Assemblies\Microsoft\Framework\.NETFramework\v4.5.1\mscorlib.dll

  Using Assembly dependency framework/System 4.0.0.0
    Source: C:\Program Files (x86)\Reference
Assemblies\Microsoft\Framework\.NETFramework\v4.5.1\System.dll

  Using Assembly dependency framework/System.Core 4.0.0.0
    Source: C:\Program Files (x86)\Reference
Assemblies\Microsoft\Framework\.NETFramework\v4.5.1\System.Core.dll

  Using Assembly dependency framework/Microsoft.CSharp 4.0.0.0
    Source: C:\Program Files (x86)\Reference
Assemblies\Microsoft\Framework\.NETFramework\v4.5.1\Microsoft.CSharp.dll


Building ClassLibrary39 for DNXCore,Version=v5.0
  Using Project dependency ClassLibrary39 1.0.0
    Source: C:\Users\davifowl\Documents\Visual Studio
14\Projects\ClassLibrary39\src\ClassLibrary39\project.json

  Using Package dependency System.Console 4.0.0-beta-22709
    Source: C:\Users\davifowl\.dnx\packages\System.Console\4.0.0-beta-22709
    File: lib\contract\System.Console.dll

  Using Package dependency System.IO 4.0.10-beta-22231
    Source: C:\Users\davifowl\.dnx\packages\System.IO\4.0.10-beta-22231
    File: lib\contract\System.IO.dll

  Using Package dependency System.Runtime 4.0.20-beta-22231
    Source: C:\Users\davifowl\.dnx\packages\System.Runtime\4.0.20-beta-22231
    File: lib\contract\System.Runtime.dll

  Using Package dependency System.Text.Encoding 4.0.10-beta-22231
    Source: C:\Users\davifowl\.dnx\packages\System.Text.Encoding\4.0.10-beta-
22231
    File: lib\contract\System.Text.Encoding.dll

  Using Package dependency System.Threading.Tasks 4.0.10-beta-22231
    Source: C:\Users\davifowl\.dnx\packages\System.Threading.Tasks\4.0.10-beta-
```

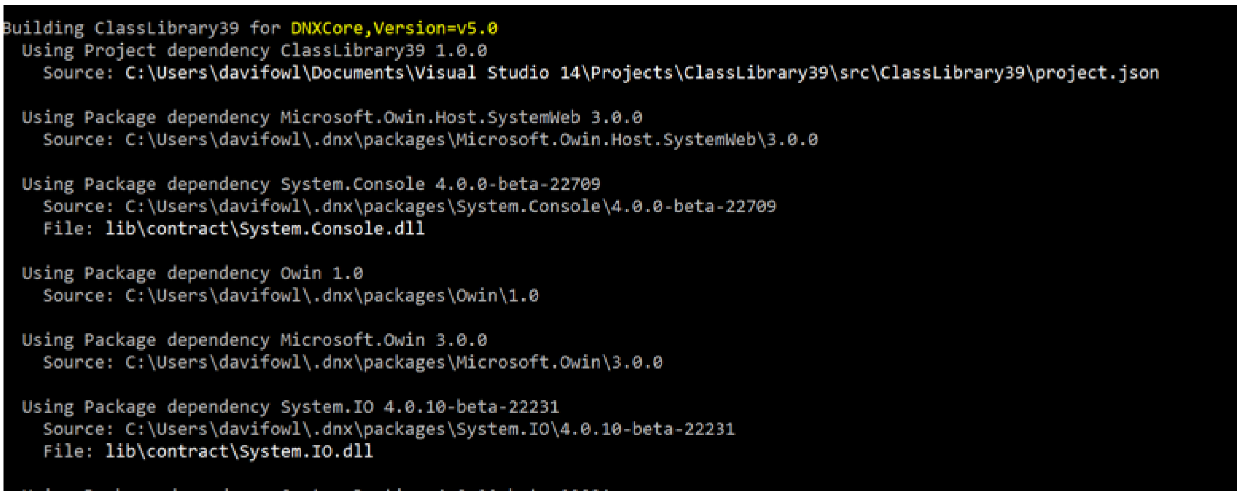
```
22231
    File: lib\contract\System.Threading.Tasks.dll
```

The output shows all of the assemblies passed into the compiler from packages and project references. When you start getting build failures, it's useful to look here to make sure that the package you are using actually works on that target platform.

Here's an example of a package that doesn't work on dnxcore50:

```
{
    "version": "1.0.0-*",
    "dependencies": {
        "Microsoft.Owin.Host.SystemWeb": "3.0.0"
    },

    "frameworks": {
        "dnx451": {
            "dependencies": {
            }
        },
        "dnxcore50": {
            "dependencies": {
                "System.Console": "4.0.0-beta-22709"
            }
        }
    }
}
```

Microsoft.Owin.Host.SystemWeb version 3.0.0 does not have any assemblies that run on dnxcore50 (take a look at the unzipped package's lib folder). When we run `kpm build`:



```
Building ClassLibrary39 for DNXCore,Version=v5.0
  Using Project dependency ClassLibrary39 1.0.0
    Source: C:\Users\davifowl\Documents\Visual Studio 14\Projects\ClassLibrary39\src\ClassLibrary39\project.json

  Using Package dependency Microsoft.Owin.Host.SystemWeb 3.0.0
    Source: C:\Users\davifowl\.dnx\packages\Microsoft.Owin.Host.SystemWeb\3.0.0

  Using Package dependency System.Console 4.0.0-beta-22709
    Source: C:\Users\davifowl\.dnx\packages\System.Console\4.0.0-beta-22709
    File: lib\contract\System.Console.dll

  Using Package dependency Owin 1.0
    Source: C:\Users\davifowl\.dnx\packages\Owin\1.0

  Using Package dependency Microsoft.Owin 3.0.0
    Source: C:\Users\davifowl\.dnx\packages\Microsoft.Owin\3.0.0

  Using Package dependency System.IO 4.0.10-beta-22231
    Source: C:\Users\davifowl\.dnx\packages\System.IO\4.0.10-beta-22231
    File: lib\contract\System.IO.dll
```

Notice it says "using Package Microsoft.Owin.Host.SystemWeb" but there is not "File:". This could be the reason for a build failure.

Share

Improve this answer

Follow

edited Oct 1 at 18:33

TylerH
21.2k ● 76 ● 79 ● 110

answered Mar 20, 2015 at 7:43

davidfowl
38.7k ● 8 ● 101 ● 114

I'm trying to use dnu list as you suggest, to determine why dnx can't resolve a dependency. But I'm getting a red "Unable to locate project.json". The assembly is in the artifacts folder, generated by checking "Produce outputs on build". Any suggestions on how to proceed?
– Mike Scott Jun 25, 2015 at 15:01

What does the artifacts folder have to do with anything? Did you reference the dependency in project.json? Is that package you're referencing available on a configured feed? – davidfowl Jun 30, 2015 at 0:28

17

I still don't know *entirely* what was wrong, but I now have a series of steps to at least make it easier to try things:

- When in doubt, reinstall dnx
  - Blowing away the package cache can be helpful
- Check `~/.config/NuGet.config` to ensure you're using the right NuGet feeds

I ended up using the following command line to test various options in a reasonably clean way:

```
rm -rf ~/.dnx/packages && rm -rf ~/.dnx/runtimes && dnvm upgrade && kpm restore
&& dnx . kestrel
```

It looks like my problem was really due to the wrong versions of the dependencies being installed. A version number of `"1.0.0-beta4"` is apparently quite different to `"1.0.0-beta4-*"`. For example, the `Kestrel` dependency installed version 1.0.0-beta4-11185 when just specified as `1.0.0-beta4`, but version 1.0.0-beta4-11262 with the `-*` at the end. I wanted to specify `beta4` explicitly to avoid accidentally using a beta3 build with the

The following project config works fine:

```
{
  "dependencies": {
    "Kestrel": "1.0.0-beta4-*",
    "Microsoft.AspNet.Diagnostics": "1.0.0-beta4-*",
    "Microsoft.AspNet.Hosting": "1.0.0-beta4-*",
    "Microsoft.AspNet.Server.WebListener": "1.0.0-beta4-*",
  },
  "commands": {
    "kestrel": "Microsoft.AspNet.Hosting --server Kestrel --server.urls
http://localhost:5004"
  },
  "frameworks": {
    "dnx451": {}
  }
}
```

Share Improve this answer Follow

answered Mar 12, 2015 at 12:41
Jon Skeet
1.5m ● 889 ● 9.3k ● 9.3k

6 This is because `-*` always gives you the latest prerelease version, while without it you get the lowest version that satisfies all dependencies (as is usual with NuGet). This test has a few examples. – Alexander Köplinger Mar 12, 2015 at 14:41

2 @AlexanderKöplinger: Thanks, that makes sense. So ...beta4 is the earliest beta4, while ...beta4-* is the latest beta4, right? – Jon Skeet Mar 12, 2015 at 14:43

4 `"frameworks": {"dnx451": {}}` had it fixed for me, no need for `dnxcore50` – vicentedealencar Jun 10, 2015 at 2:50

Your first command also helped me to get past being stuck on the beta5 version. I tried running `dnvm upgrade-self`, this would not upgrade to the latest version. Running the VS command prompt as admin showed dnvm version as `rc1...`, however when not as admin it was `beta5...`. After your command both admin and non admin command prompts showed as the `rc2...` (latest) version. – JabberwockyDecompiler Nov 2, 2015 at 16:05

For those using mono and wondering whether to choose `dnx451` or `dnxcore50` this answer helped me understand this topic a bit more: stackoverflow.com/a/30846048/89590 Short answer: `dnx451` is appropriate for mono. – Nate Cook Dec 11, 2015 at 18:39

8

You can set an env var named `DNX_TRACE` to `1` to see a TON more diagnostic info. Be warned, it's *a lot* more info!

Share Improve this answer Follow

answered Mar 13, 2015 at 0:30
Eilon
25.7k ● 3 ● 86 ● 105

@JonSkeet BTW the other answers (including your self-answer) contain great info about diagnosing and repairing the specific problem you encountered. I kept this answer super brief because it's just another different answer that could lead to more clues as to why the problem happened in the first place. – Eilon Mar 13, 2015 at 6:54

Absolutely - I appreciate that :) – Jon Skeet Mar 13, 2015 at 6:56

To get it to work I modified my `project.json` .. it now looks like:

3

```
{
"dependencies": {
    "Kestrel": "1.0.0-*",
    "Microsoft.AspNet.Diagnostics": "1.0.0-*",
    "Microsoft.AspNet.Hosting": "1.0.0-*",
    "Microsoft.AspNet.Server.WebListener": "1.0.0-*",
    "Microsoft.AspNet.StaticFiles": "1.0.0-*"
},
"commands": {
    "web": "Microsoft.AspNet.Hosting --server
Microsoft.AspNet.Server.WebListener --server.urls http://localhost:5001",
    "kestrel": "Microsoft.AspNet.Hosting --server Kestrel --server.urls
http://localhost:5004"
},
"frameworks": {
    }
}
```

The key seemed to be the frameworks section.

Also the rename changed how `k web` works so that its now `dnx . web` or `dnx . kestrel`

Update - bit more info

Oddly, after running with no frameworks defined it went and got a bunch of extra stuff when I did `kpm restore` :

```
...
Installing Microsoft.Framework.Logging 1.0.0-beta4-11001
Installing Microsoft.Framework.Logging.Interfaces 1.0.0-beta4-11001
Installing Microsoft.Framework.DependencyInjection.Interfaces 1.0.0-beta4-11010
Installing Microsoft.Framework.DependencyInjection 1.0.0-beta4-11010
Installing Microsoft.Framework.ConfigurationModel 1.0.0-beta4-10976
Installing Microsoft.Framework.ConfigurationModel.Interfaces 1.0.0-beta4-10976
Installing Microsoft.AspNet.Hosting.Interfaces 1.0.0-beta4-11328
Installing Microsoft.AspNet.FeatureModel 1.0.0-beta4-11104
Installing Microsoft.AspNet.Http 1.0.0-beta4-11104
Installing Microsoft.AspNet.FileProviders.Interfaces 1.0.0-beta4-11006
Installing Microsoft.Framework.Caching.Interfaces 1.0.0-beta4-10981
Installing Microsoft.AspNet.FileProviders 1.0.0-beta4-11006
Installing Microsoft.AspNet.Http.Core 1.0.0-beta4-11104
Installing Microsoft.AspNet.WebUtilities 1.0.0-beta4-11104
Installing Microsoft.Net.Http.Headers 1.0.0-beta4-11104
Installing Microsoft.AspNet.Http.Interfaces 1.0.0-beta4-11104
Installing Microsoft.Framework.Runtime.Interfaces 1.0.0-beta4-11257
Installing Microsoft.AspNet.Server.Kestrel 1.0.0-beta4-11262
Installing Microsoft.Net.Http.Server 1.0.0-beta4-11698
Installing Microsoft.Net.WebSockets 1.0.0-beta4-11698
Installing Microsoft.Net.WebSocketAbstractions 1.0.0-beta4-10915
Installing Microsoft.Framework.WebEncoders 1.0.0-beta4-11104
Installing Microsoft.Framework.OptionsModel 1.0.0-beta4-10984
Installing Microsoft.AspNet.Http.Extensions 1.0.0-beta4-11104
Installing Microsoft.AspNet.Diagnostics.Interfaces 1.0.0-beta4-12451
Installing Microsoft.AspNet.RequestContainer 1.0.0-beta4-11328
```

.. then it ran fine. Then I switched back in the framework section

```
"frameworks": {
    "dnx451": {}
}
```

.. and it still worked, whereas before it would throw up an error !

Very odd!

(Im running `1.0.0-beta4-11257`)

Further update

I spun up a new Ubuntu instance, and got the same error as you .. My thought was that the issue may be caused by it only trying to get packages from `nuget.org` and not `myget.org` (which has the newer things) so i dropped in a `NuGet.Config` into the root of the project..

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <packageSources>
    <add key="AspNetVNext" value="https://www.myget.org/F/aspnetvnext/" />
    <add key="NuGet" value="https://nuget.org/api/v2/" />
  </packageSources>
</configuration>
```

.. this seems to have fixed it for me by getting the correct versions (after another `kpm restore`).

Share

Improve this answer

Follow

edited Mar 12, 2015 at 12:31

answered Mar 12, 2015 at 11:20

Stephen Pope
3,551 ● 1 ● 19 ● 21

1 Re the "dnx . kestrel" part - indeed, hence the command I showed :) With that config, I get a different error: System.TypeLoadException: Could not load type 'Microsoft.Framework.DependencyInjection.LoggingServiceCollectionExtensions' from assembly 'Microsoft.Framework.Logging, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null'. Which version of DNX are you using? – Jon Skeet Mar 12, 2015 at 11:25

1 When i did "dnx . web" the first time i got: `System.InvalidOperationException: Failed to resolve the following dependencies for target framework 'DNX,Version=v4.5.1' and it suggested a list of the things it was missing. – Stephen Pope Mar 12, 2015 at 11:37

Interesting. What platform is this on, btw? – Jon Skeet Mar 12, 2015 at 11:38

Did you do 'source ~/.bashrc' to reload the environment variables after you upgraded DNX ? Also I had to do "dnvm upgrade" + "dnvm use default" – Stephen Pope Mar 12, 2015 at 11:39

DNX hadn't updated by .bashrc... possibly because I built it manually yesterday. Will try using the updated instructions instead... – Jon Skeet Mar 12, 2015 at 11:40

2

These days, all of my `package.json` versions end in `"-rc2-*"`

(Only exceptions I've seen so far are the `Microsoft.Framework.Configuration` packages, which need to be either `"1.0.0-rc1-*"` or `"1.0.0-*"`)

Regarding the "version trains" that @davidfowl mentions, it seems a LOT of pain has disappeared between beta8 and rc2.

```
dnvm upgrade -u -arch x64 -r coreclr
```

I've had the most luck on `coreclr` with these 2 NuGet feeds:

```
"https://www.myget.org/F/aspnetvnext/"
"https://nuget.org/api/v2/"
```

When I *do* have missing package problems, 90% of the time it's these same culprits:

```
Newtonsoft.Json
Ix-Async
Remotion.Linq
```

Most of the time, I can get around these by forcing the main NuGet.org feed:

```
dnu restore;
dnu restore -s https://nuget.org/api/v2
```

Here is my working config.json:

```
{
"dependencies": {
    "Microsoft.AspNet.Diagnostics": "1.0.0-rc2-*",
    "Microsoft.AspNet.Diagnostics.Entity": "7.0.0-rc2-*",
    "Microsoft.AspNet.Hosting": "1.0.0-rc2-*",
    "Microsoft.AspNet.Http": "1.0.0-rc2-*",
    "Microsoft.AspNet.Http.Abstractions": "1.0.0-rc2-*",
    "Microsoft.AspNet.Mvc.Core": "6.0.0-rc2-*",
    "Microsoft.AspNet.Mvc.Razor": "6.0.0-rc2-*",
    "Microsoft.AspNet.Owin": "1.0.0-rc2-*",
    "Microsoft.AspNet.Routing": "1.0.0-rc2-*",
    "Microsoft.AspNet.Server.Kestrel": "1.0.0-rc2-*",
    "Microsoft.AspNet.Server.WebListener": "1.0.0-rc2-*",
    "Microsoft.AspNet.Session": "1.0.0-rc2-*",
    "Microsoft.AspNet.StaticFiles": "1.0.0-rc2-*",
    "EntityFramework.Commands": "7.0.0-rc2-*",
    "EntityFramework.Core": "7.0.0-rc2-*",
    "EntityFramework.InMemory": "7.0.0-rc2-*",
    "EntityFramework.MicrosoftSqlServer": "7.0.0-rc2-*",
    "EntityFramework.MicrosoftSqlServer.Design": "7.0.0-rc2-*",
    "EntityFramework.Relational": "7.0.0-rc2-*",
    "EntityFramework7.Npgsql": "3.1.0-beta8-2",
    "Microsoft.Extensions.Logging.Abstractions": "1.0.0-rc2-*",
    "Microsoft.Extensions.Logging.Console": "1.0.0-rc2-*",
    "Microsoft.Extensions.DependencyInjection": "1.0.0-rc2-*",
    "Microsoft.Extensions.DependencyInjection.Abstractions": "1.0.0-rc2-*",
    "Microsoft.Framework.Configuration.CommandLine": "1.0.0-*",
    "Microsoft.Framework.Configuration.EnvironmentVariables": "1.0.0-*",
    "Microsoft.Framework.Configuration.Json": "1.0.0-*"
},
"commands": {
    "ef": "EntityFramework.Commands",
    "dev": "Microsoft.AspNet.Hosting --ASPNET_ENV Development --server
Microsoft.AspNet.Server.Kestrel --server.urls http://localhost:5004"
},
"frameworks": {
    "dnxcore50": {}
}
}
```

Share

Improve this answer

Follow

edited Nov 16, 2015 at 19:56

answered Nov 16, 2015 at 18:20

CrazyPyro

3,587 ● 3 ● 32 ● 39

The above list is not from config.json but rather project.json but I still upvoted because the list provided me with useful dependencies that I didn't previously know about. – RonC Apr 5, 2016 at 15:57

1

I was having dependency missing issues as well with trying to appease dnxcore50 and dnx451 references.

If I understand this right "dependencies": {} are shared between the frameworks.

Then "dependencies":{} within the "frameworks": are specific to that framework.

dnxcore50 is a modular runtime (self contained) so it basically contains all the core runtimes need to run a program unlike classic .net framework where you have core dependencies scattered about elsewhere.

So with that said I wanted to stick to the minimal approach incase I decided to host on mac or linux at some point.

Update Ran into weird dependency issues with cshtml views, just went with dnx451 for now.

This is my project.json

```
{
"webroot": "wwwroot",
"version": "1.0.0-*",

"dependencies": {
    "System.Runtime": "4.0.10",
    "Microsoft.AspNet.Hosting": "1.0.0-beta4",
    "Microsoft.AspNet.Mvc": "6.0.0-beta4",
    "Microsoft.AspNet.Server.IIS": "1.0.0-beta6-12075",
    "Microsoft.AspNet.Server.WebListener": "1.0.0-beta6-12457",
    "Microsoft.Framework.DependencyInjection": "1.0.0-beta4",
    "Microsoft.Framework.DependencyInjection.Interfaces": "1.0.0-beta5"
 },

"commands": {
"web": "Microsoft.AspNet.Hosting --server Microsoft.AspNet.Server.WebListener -
-server.urls http://admin.heartlegacylocal.com"  },

"frameworks": {
"dnx451": { }
 }
},

"publishExclude": [
"node_modules",
"bower_components",
"**.xproj",
"**.user",
"**.vspscc"
],
"exclude": [
  "wwwroot",
  "node_modules",
  "bower_components"
  ]
}
```

Share

Improve this answer

Follow

edited Jul 16, 2015 at 19:52

answered Jul 16, 2015 at 16:32

dynamiclynk

2,331 ● 28 ● 31