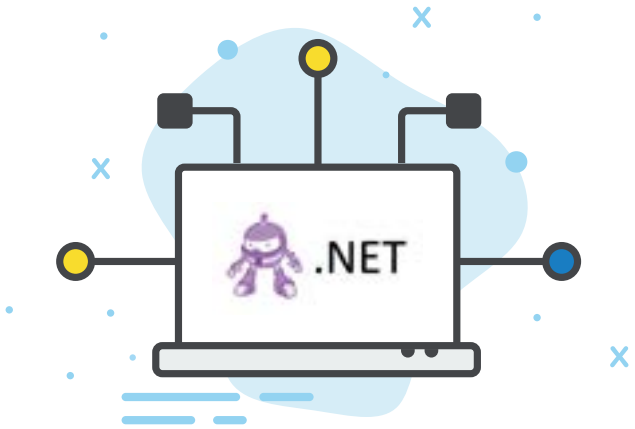
Headspring

# Tiny steps: A strategy for moving to .NET 5

**With the impending arrival of .NET 5, your applications are suddenly at risk of dying on the vine. The good news is you're probably part of the way there. You can start thinking about previous versions of .NET Core as beta releases of .NET 5. So if you've already made it through previous Microsoft updates you're on the right path.**




**What you think of as this:**

```
.NET 4.x ----------------------x
                               .NET 5--->
.NET Core ---1.0---2.0---3.0---x
```

**Is, essentially, this:**

```
.NET 4.x ----------------------x
                               .NET 5--->
.NET 5 --Beta 1--Beta 2--Beta 3-
```

Thus, we can use .NET Core / .NET Standard as a migration tool.

Regardless of where you are on the spectrum now—from .NET Framework to any version of ".NET 5 Beta"—porting your apps to .NET 5 proper may feel like moving an entire city. You're not wrong. But there is a right (or at least smart) way to do it.

From-the-ground-up rewrites are risky and disruptive. You can't build a new city on the side and expect everyone to move in next Tuesday. What you can do is move your "city" piece-by-piece, putting new things in place to deal with modern realities.

So how do we actually tackle this Herculean effort in our own applications? Implementations will vary, so let's talk strategy. Taking a hypothetical application as an example, this playbook will show you how to take tiny steps that will get you from .NET beta to .NET 5-ready in time for the release.

*"NET Core is just the early adopter's risk mitigation strategy for getting us over the hurdle of .NET 4 to .NET 5 this year."*

# Steps for migrating your existing projects

Let's assume you're on a typical legacy application with slightly-old everything:

- VS2017
- .NET 4.6.1
- ASP.NET MVC 5
- JSON.NET 8.x
- Entity Framework
- xUnit + Should
- PowerShell build script invoking NuGet, MSBuild, xUnit console, etc.
- 1 web app project, 30-some tangled class library projects

*Take tiny steps and perform experiments in isolation*

*"There's no need to stop delivering value if you take tiny steps throughout the year."*

**STEP 1** Upgrade to VS2019

**STEP 2** Upgrade to .NET Framework 4.8

**STEP 3** **Update NuGet packages to those cross-targeting .NET Core/Standard. Upgrade JSON.NET to the latest version.**

**STEP 4** **Replace Nuget packages that have no modern support. E.g., I replaced the Should library with Shouldly.**

**STEP 5** **Switch from EF to EF Core. Heavy testing is critical here.**

**STEP 6** **Switch to modern csproj on the old .NET Framework. How? Experiment, then apply!**

**Experiment #1: Modern csproj on old .NET Framework (scorched earth)**

- Create an empty folder File / New Solution
- Create Modern Empty Projects with the same names as projects for the real solution, but no code
- Restore the Interproject References (no code)
- Restore the NuGet References (no code)
- Start Bulk Copying files from the original into the experiment

Now, it's time to take those learnings and apply them to the real, in-place work. Each of your 30-some projects should run through these sub-steps before you move on to the next project.

**In-place Upgrades: Modern csproj on old .NET Framework (realistic)**

- Delete one project and do a Git commit
- Create a File \ New Project with the same name
- Reestablish Interproject References (commit)
- Reestablish NuGet packages (commit)
- Bulk Copy code files, see what red flags appear, fix up C# Files
- Repeat. Do one project at a time, until eventually, everything is on the new format.

STEP 7

**Now we're as up-to-date as possible on the regular framework. It's time to upgrade ASP.NET to ASP.NET Core 2.**

*First, run an experiment first just to see what's different between them*

**Experiment #2: ASP.NET to ASP.NET Core 2**

- Go to File \ New Project for a basic "Hello World" ASP.NET Core 2 web application
- Study the new concepts in Main( ) and Startup.cs
- Compare with your existing application's own start-up code
- Learn and apply

You'll end up with an application that is just as good as something on .NET Core but is still compiling against the old .NET Framework. All you'll need is to go into the csproj and retarget .NET Framework to .NET Core.

STEP 8

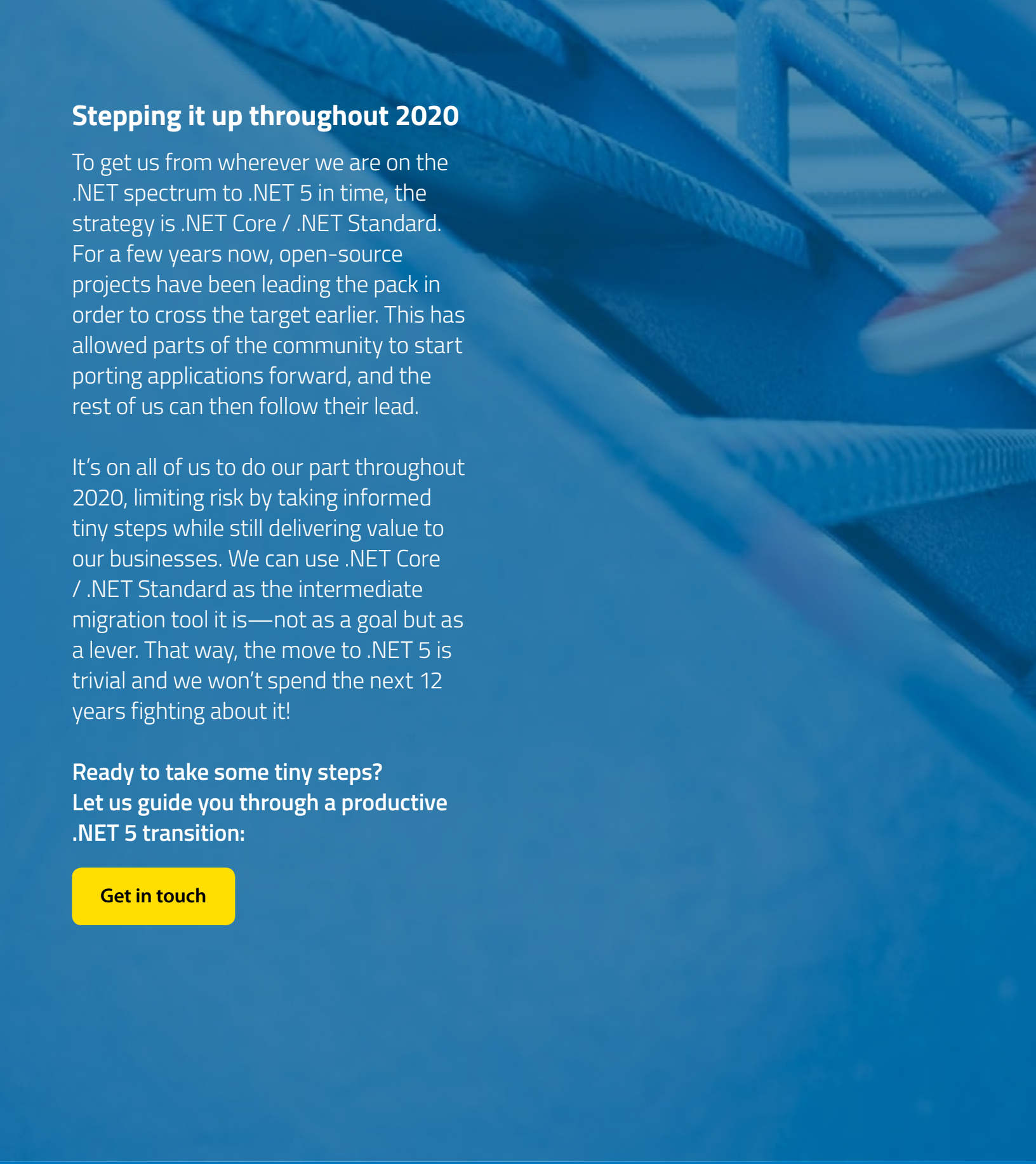
**Now you can move from ASP.NET Core 2 to ASP.NET Core 3. How do you do it, or know what has changed?**

**Experiment #3: ASP.NET Core 2 to 3**

- Create 2 Empty Folders
- Go to File \ New Project for ASP.NET Core 2
- Go to File \ New Project for ASP.NET Core 3
- Use your Diff tool between the two folders to learn the fundamental differences in Main, Startup, and the web application's csproj

Now we can (easily ) move from .NET Core 3 to .NET 5 when it arrives.

*Keep running experiments when you get there to apply changes in a targeted way.*



## Stepping it up throughout 2020

To get us from wherever we are on the .NET spectrum to .NET 5 in time, the strategy is .NET Core / .NET Standard. For a few years now, open-source projects have been leading the pack in order to cross the target earlier. This has allowed parts of the community to start porting applications forward, and the rest of us can then follow their lead.

It's on all of us to do our part throughout 2020, limiting risk by taking informed tiny steps while still delivering value to our businesses. We can use .NET Core / .NET Standard as the intermediate migration tool it is—not as a goal but as a lever. That way, the move to .NET 5 is trivial and we won't spend the next 12 years fighting about it!

**Ready to take some tiny steps?**
**Let us guide you through a productive .NET 5 transition:**

**Get in touch**

Headspring

10415 Morado Circle #300, Austin, TX 78759
info@headspring.com | (877) 459-2260 | www.headspring.com