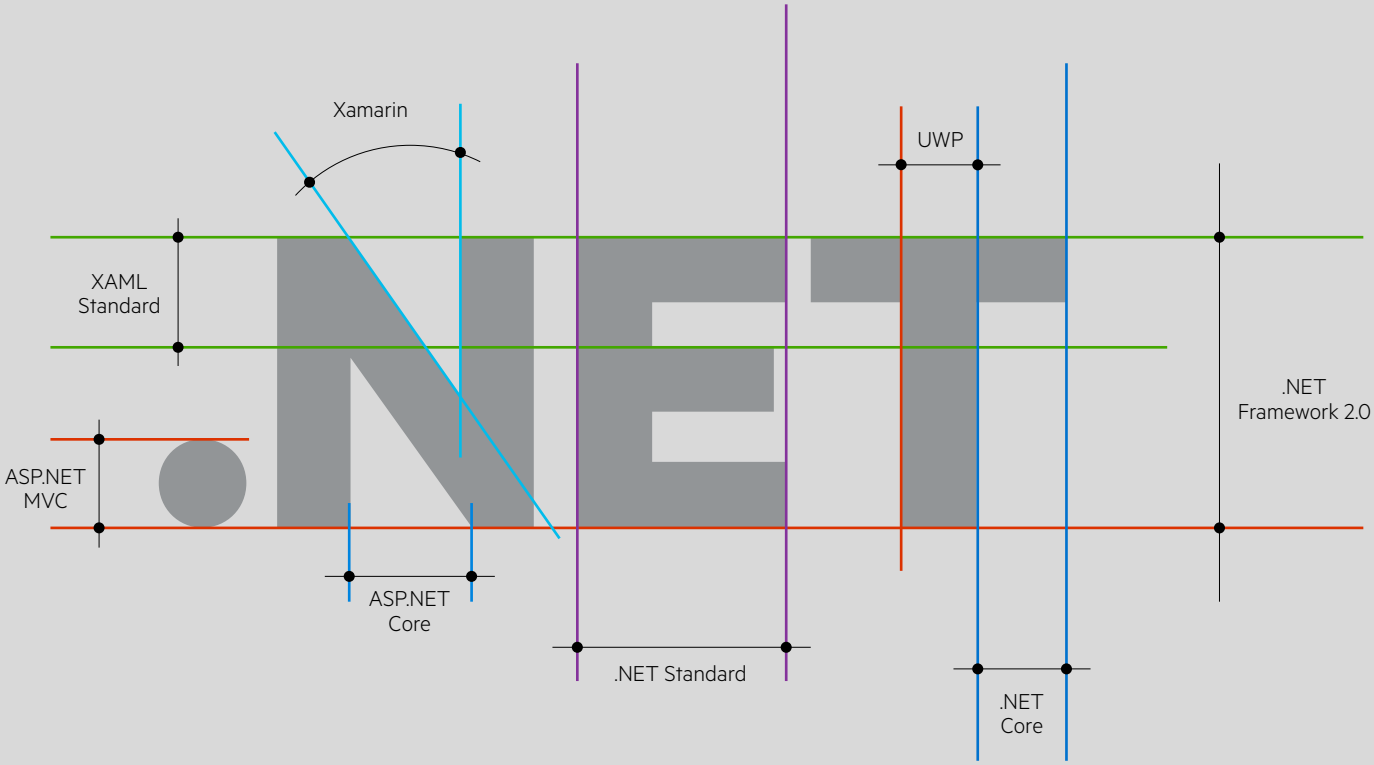
Progress®



The State of .NET in 2018

How the New .NET Standard is Making You a Better Developer

WHITEPAPER

Table of Contents

It's a rare luxury to stand atop years of success, only to re-shape what the future holds. That is exactly where .NET finds itself—relishing years of developer loyalty, while pivoting itself to a position for upcoming success. At this key inflection point in the history of .NET, some specific focus areas define the gold standard in .NET development going forward. This whitepaper examines the new capabilities of .NET—including fundamental improvements and standardization efforts surrounding it—to bring forth the best ideas for modern .NET development.

State of .NET

The very first beta of the .NET Framework was released 17 years ago. The .NET Framework—originally called 'Next Generation Windows Services' (NGWS)—was an attempt to combine the APIs and system level abstractions that most Windows applications needed. The next two decades saw the .NET Framework flourish by leaps and bounds, with almost no other development technology garnering the kind of success, loyalty, love and growth it's enjoyed.

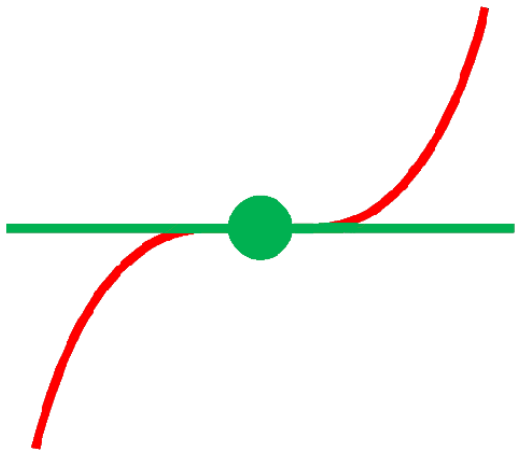
With time, the .NET Framework grew in features and more abstractions that make life easy for developers. The developer community stepped up to build a rich ecosystem around it, resulting in fantastic tools, language support, and a mature framework to build Windows apps on. Enterprises, large and small, bought into the .NET promise and made big bets with apps built on top of the .NET technology stack. Developers flourished in the .NET ecosystem, with ever more sophisticated tools to aid in building polished apps.

Why Pivot?

With all the advancements over the last 15 years, the .NET Framework grew and grew. This was very convenient for developers, but it represented a giant framework nonetheless. This presented a new challenge since various apps across web/desktop/mobile platforms started depending on the same foundation. Updates became increasingly difficult and the ecosystem started becoming fragile. It was clearly time for a change.

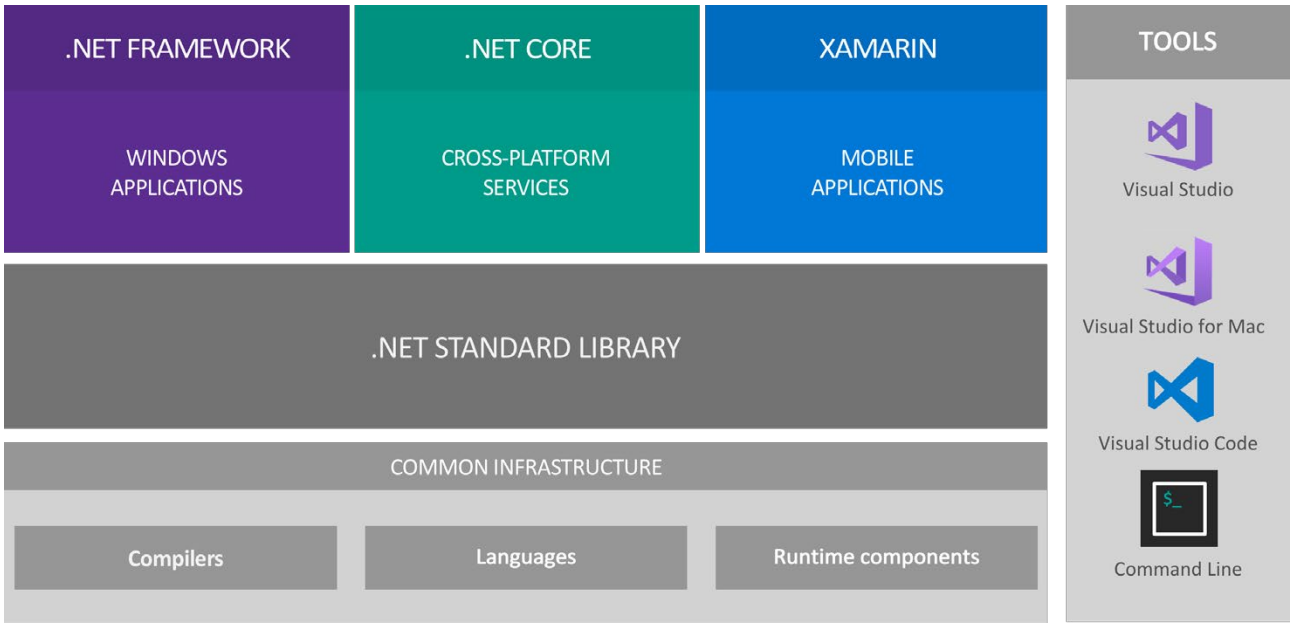
A lot has changed in the software industry during the lifetime of .NET. The mobile revolution has changed the way we interact with software, cloud computing has turned hardware infrastructure into a service and cheaper computing has led to exponential growth of technology. Machine learning is paving the way for software intelligence and portability across platforms is of utmost importance. We truly live in a "cloud-first" and "mobile-first" world now.

With this new era of computing, .NET had to evolve to support the next generation of apps, stepping out of just Windows, support multiple platforms and focus on portability. For .NET to remain relevant for developers, it had to be reinvented—a critical pivot that will set up .NET for success over the next decade. In differential calculus, an inflection point is a point on a curve at which the curve changes from being concave to convex. .NET was at its inflection point, setting it up nicely for the future.



Behold the New

Let's look at the bigger picture, as we stand today.



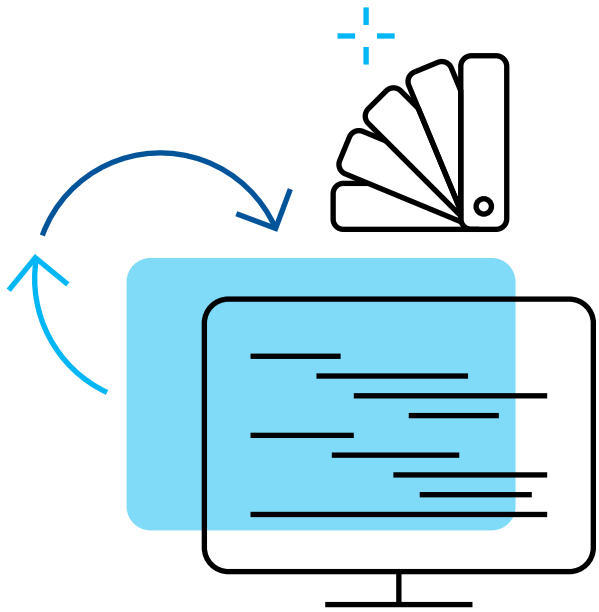
Source: Introducing .NET Standard

This isn't your grandma's .NET or your grandpa's tooling. No offense to grandparents, but things have changed a lot in the .NET landscape. There are several flavors of .NET now—.NET Framework, .NET Core, Mono, Unity and other Base Class Libraries (BCLs)—all catering specific app platforms towards the developer's benefit.

The giant monolithic .NET Framework still exists and still runs all Windows desktop, web and mobile apps. .NET Core is the new kid on the block, written from the ground up, lean, modular, and sporting a cross-platform Common Language Runtime (CLR). The implications of .NET Core are huge. For the first time, .NET apps can run natively outside of Windows, on Mac and Linux. Xamarin apps still run on Mono—which is a long-standing open source port of .NET to other platforms. With today's .NET, developers can target virtually any app platform and any device family. The recent release of .NET Core 2.0 has big implications for the future of .NET and tooling going forward.

Along with the changes in .NET offerings, .NET development tools have also evolved. Visual Studio has various flavors and lowers the barrier to entry with a more "come-as-you-are" mindset. On Windows, Visual Studio has become synonymous with .NET development—the one IDE where developers often spend their entire day. Visual Studio has come a long way as an IDE—from a giant behemoth to a carefully crafted and streamlined IDE, catering to specific development workflows. The Visual Studio feature-richness continues though—it's the ubiquitous IDE to build modern web, desktop, mobile, cloud and future-facing solutions.

A few years back, you would probably laugh if someone told you that Visual Studio would run natively on a Mac but that's the reality today. Visual Studio for Mac is a full-featured native IDE and brings most of the VS development comforts over to MacOS for modern mobile, web and cloud development. Then there is Visual Studio Code—a beautiful light-weight truly cross-platform code editor. Modern .NET developers aren't shy of using the command line either—the one thing common across all development platforms. And .NET obliges by providing solid .NET CLI tooling for consistent cross-platform usage.



Why Standards?

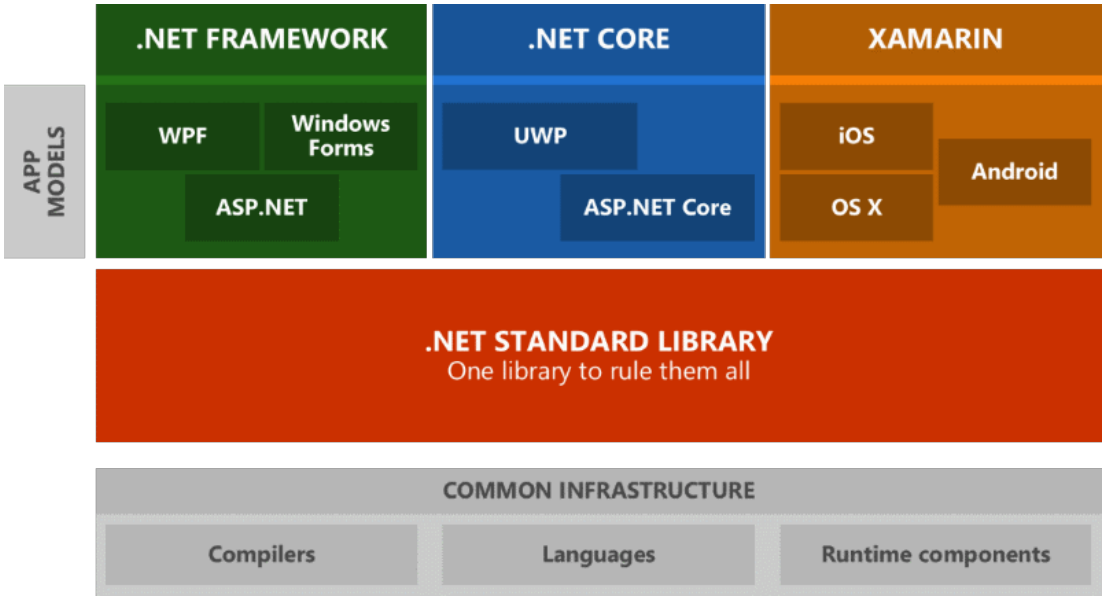
Most .NET developers are likely not doing just one type of development. There may be some desktop development through WinForms/WPF, or modern web development with ASP.NET Core, or cross-platform mobile apps with Xamarin. Wouldn't it be nice if platform-independent code artifacts could be shared across various .NET platforms?

.NET, as we mentioned, comes in several flavors now—.NET Framework, .NET Core, Mono, Unity and other BCLs—all catering to specific app platforms. Customized .NET flavors are great, but come with the obvious risk of fragmentation. To .NET developers, this should all be transparent—there is an obvious need for unification and code reusability across various .NETs. There is also some need to unify XAML, the UI markup language used across various .NET platforms.

Let's talk standardization!

.NET Standard

.NET Standard is a formal specification of .NET APIs that is intended to be available on all supporting .NET implementations as a way of conformity. The motivation behind .NET Standard is simple: have greater uniformity in the .NET ecosystem and allow for code portability/reuse.



Source: Introducing .NET Standard

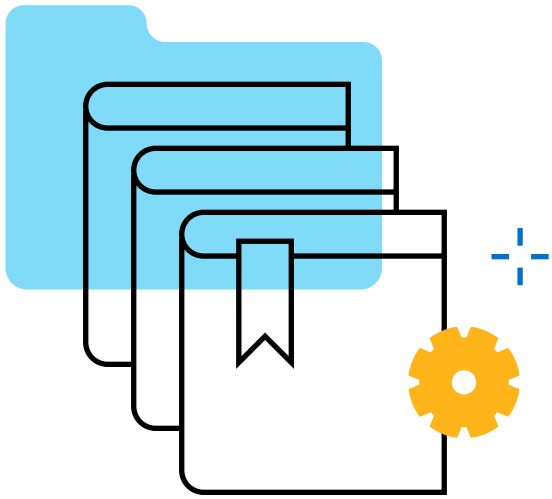
You may ask, "What about Portable Class Libraries (or PCLs)?" PCLs had the same promise of portability across platforms, however, implementations suffered as the number of PCL profiles increased. Developers were made to choose device families and pick the lowest common denominator across supported APIs.

.NET Standard is the next generation idea on portability, putting the API implementation responsibilities back on .NET platforms. While PCLs let developers code to the lowest common denominator of APIs available in supported platforms, .NET Standard takes a different route. The onus is now on the app platforms to implement .NET APIs. When your app supports a certain .NET Standard, you are guaranteed all APIs in that version of .NET Standard to be supported in all supporting app platforms. The goal of .NET Standard is two-fold:

- Define a uniform set of Base Class Library (BCL) APIs across all implementations of .NET, irrespective of workload or platform execution environment

- Enable developers to write portable libraries that are usable across .NET implementations, using the defined set of APIs

The various .NET implementations target specific versions of .NET Standard. Each .NET version essentially advertises the highest .NET Standard version it supports, which also means it supports previous versions. With the release of .NET Standard 2.0, .NET Core gets a lot of the feature parity with the full .NET Framework in terms of APIs.



Better Portability

You can get a glimpse of the different implementations of .NET and their support of corresponding .NET Standard versions—things are moving along nicely. All this means more portability of .NET code and libraries for developers—reuse all things from a developer's perspective.

.NET Standard	1.0	1.1	1.2	1.3	1.4	1.5	1.6	2.0
.NET Core	1.0	1.0	1.0	1.0	1.0	1.0	**1.0**	**2.0**
.NET Framework	4.5	**4.5**	**4.5.1**	**4.6**	4.6.1	4.6.1	4.6.1	**4.6.1**
Mono	4.6	4.6	4.6	4.6	4.6	4.6	**4.6**	**5.4**
Xamarin.iOS	10.0	10.0	10.0	10.0	**10.0**	10.0	**10.0**	**10.14**
Xamarin.Mac	3.0	3.0	3.0	3.0	3.0	3.0	**3.0**	**3.8**
Xamarin.Android	7.0	7.0	7.0	7.0	7.0	7.0	**7.0**	**8.0**
Universal Windows Platform	10.0	10.0	10.0	10.0	10.0	vNext	vNext	**vNext**
Windows	8.0	**8.0**	**8.1**					
Windows Phone	8.1	8.1	**8.1**					
Windows Phone Silverlight	**8.0**							

Source: .NET Standard Versions

Note: **Bold text** indicates when a .NET implementation added support for a given .NET Standard version.

The common APIs for the .NET Framework and Xamarin were used as a basis for .NET Standard. With .NET Standard 2.0, this surface extends across the .NET ecosystem forming a Base Class Library of over 32,000 APIs. These additions make it much easier to port existing code over to .NET Standard and there are API analyzers to help developers bring their code over to be .NET Standards compliant. Since .NET Standard acts as a BCL API definition, additional Framework Class Libraries can provide platform specific functionality on top of their support for a specific .NET Standard version. Because of this, moving existing applications to frameworks that use .NET Standard implementations gets much easier.

.NET Standard Dependency

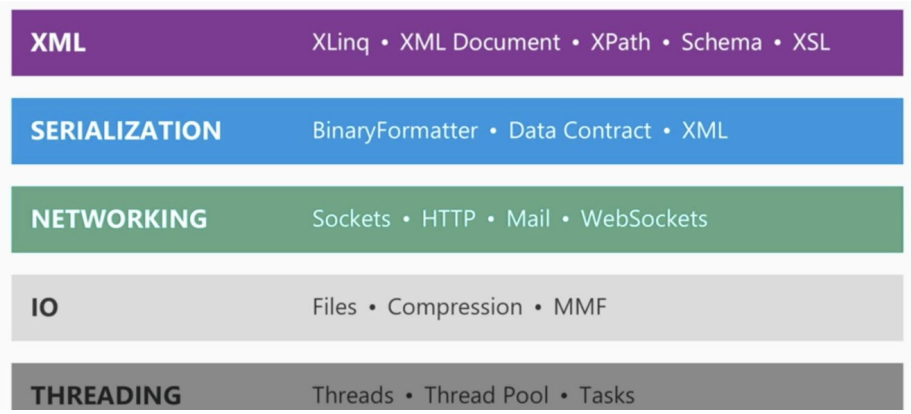
A developer's time is valuable and maintaining long lists of APIs or profiles to adhere to .NET Standard isn't efficient. Managing .NET Standard compliance in a project is extremely simple. Instead of requiring the individual dependencies that make up .NET Standard, a single reference is used.

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>netstandard2.0</TargetFramework>
  </PropertyGroup>
</Project>
```

The single reference to *<TargetFramework>netstandard2.0</TargetFramework>* in a .csproj file provides the API definitions needed for your application. This works because the reference to netstandard is a meta-package. The netstandard package does not contain any dlls, but instead references other packages required by the application. Only packages that satisfy the .NET Standard implementation for the platform your application targets will be added to the project. Therefore, the same netstandard reference is applicable to Xamarin, ASP.NET Core and the like.

.NET Core 2.0

With the addition of both new and backward-compatible features, .NET Core 2.0 represents a significant release for Microsoft. Only 43 .NET APIs are not supported. This makes .NET Core 2.0 almost at feature parity with .NET Framework 4.6.1. The reasons to not upgrade to .NET Core 2.0 are getting fewer.



Source: Introducing .NET Standard

With .NET Core 2.0 and .NET Standard 2.0, there's also little need to worry about legacy code. There are several attributes to .NET Core that make it well suited for transitioning from legacy code. Because .NET Standard 2.0 is at near API parity with .NET Framework 4.6.1, most applications can make use of code they already have. In addition, .NET Core includes a compatibility shim that allows projects to reference existing .NET Framework NuGet packages and projects. This means most packages on NuGet today are already compatible with .NET Core 2.x without the need to be recompiled. Having the ability to migrate existing applications to the next generation of .NET gives developers the freedom to write future-proof code and move the business forward with minimal risk.

XAML Standard

At its core, XAML is just a simple XML-based markup language. There are no hardcore syntax specifications and no one team at Microsoft really owns XAML. As a result, XAML morphed over time as more app platforms applied it as the UI layer. After all, it was up to the app platform's rendering engine to make sense of the XAML markup—so each one went a little in their own direction for maximum benefit.

All this brings us to today's fragmented XAML world. WPF, Silverlight, UWP and Xamarin.Forms all talk a slightly different dialect of XAML. Developers still love XAML and the core concepts along with tooling remain about the same—there are just those nagging differences in XAML markup across platforms.

Consider the two most modern app platforms that use XAML—UWP and Xamarin. Forms. To do the exact same UI renderings, developers need to use different XAML markup, as illustrated in the handful of examples below:

- Placeholder control: StackLayout vs StackPanel?
- Text field control: Label vs TextBlock?
- Text entry control: Entry vs TextBox?
- Button control name: Text vs Content?
- ForeColor property: TextColor vs ForeGround?

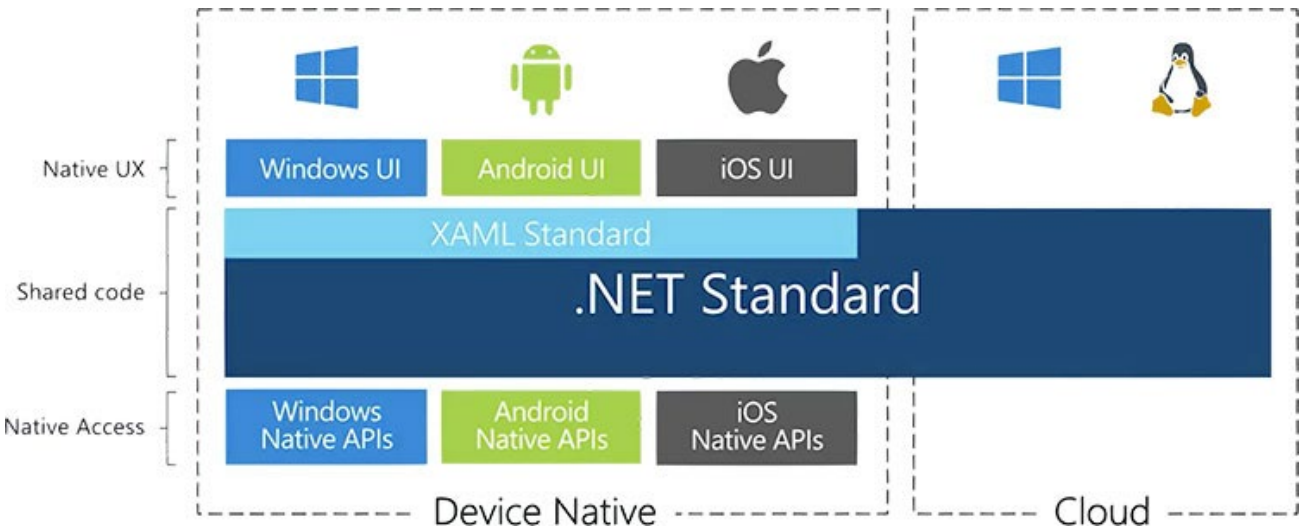
This XAML fragmentation impedes code portability and is just frustrating for developers. Clearly we can do better.

XAML Standard is a standards-based effort to unify XAML dialects across app platforms. The goal is to specify a standard XAML vocabulary and have all supporting app platforms adhere to the standard, so apps should be able to share common XAML-based UI definitions.

These are still early days for XAML Standard. The XAML Standard specification is being developed out in the open with collaboration from the developer community. Post-specification, the immediate plan is to support the UWP and Xamarin.Forms platforms. Developers can continue developing UWP/Xamarin.Forms apps as they do today. When XAML Standard support is enabled, the XAML markup will just be reusable and shared between the frameworks.

Ever since the Microsoft's acquisition of Xamarin, hardcore XAML fans have been hoping for a dialect of XAML that lets them target all app platforms—not just Windows, but iOS and Android as well. The dream has been to re-define the Universal in UWP to include non-Microsoft platforms as well. XAML Standard is a step in the right direction.

It should be noted that XAML Standard is a UI markup specification. Under the covers, it will use the native UI rendering mechanism of the supporting app platform, thus not holding back the developer or platforms in any way. Architecturally, you can see where the new XAML Standard, along with .NET Standard specifications, fit in the Microsoft development stack. It's critical to the true cross-platform portability story.



Source: Microsoft

For XAML Standard V1, the goal is to come up with a list of commonly used UI components and standardize their usage through specified properties/methods/events. The point is, no matter what platform you are writing apps for, you will set up the UI consistently using the same XAML syntax. Here's the list of some of the popular UI controls that are being standardized for properties and events:

- Button
- TextBlock
- TextBox
- ComboBox
- Grid
- StackPanel
- Page
- UserControl

Based on the type of control, developers will have standard ways of setting the content inside the UI component. Controls like Button, TextBlock, TextBox, and ComboBox deal with textual content and often need developer control over how the text is rendered. These controls will sport the following standard properties around managing Font:

- FontSize
- FontWeight
- FontStyle
- FontFamily

Developers often need to control the size and placement of common UI controls. To further aid standardization, the following properties will be available on all the UI controls listed above:

- Margin
- HorizontalAlignment
- VerticalAlignment
- Height
- Width

Aside from app developers not having to remember multiple dialects of XAML for different platforms, the XAML Standards spec has one huge potential—portability. How often do you have a piece of UI that is very similar between the same app on various platforms?

Consider a user settings dialog or a credit card payment screen. Wouldn't it be nice if the UI could be shared?

Sharing of UI across platforms is exactly what the XAML Standard enables. It allows you to apply the same syntax across all supported platforms. This enables you to bundle up commonly-used piece of UI using the standardized controls into a single XAML page. When this happens, your XAML page becomes shareable and you can take it to any platform and use it as is without modification. Imagine the kind of reusability this will bring to your code bases supporting apps on different platforms! All of this is possible with the XAML Standard.

Better Web Apps with ASP.NET Core

Also fresh off the press is ASP.NET Core 2.0 with full support for .NET Core 2.0 and lots of tooling enhancements. ASP.NET Core 2.X provides .NET developers all the tooling and framework features needed to build modern rich web applications. Here are some things to get excited about with ASP.NET Core 2.0:

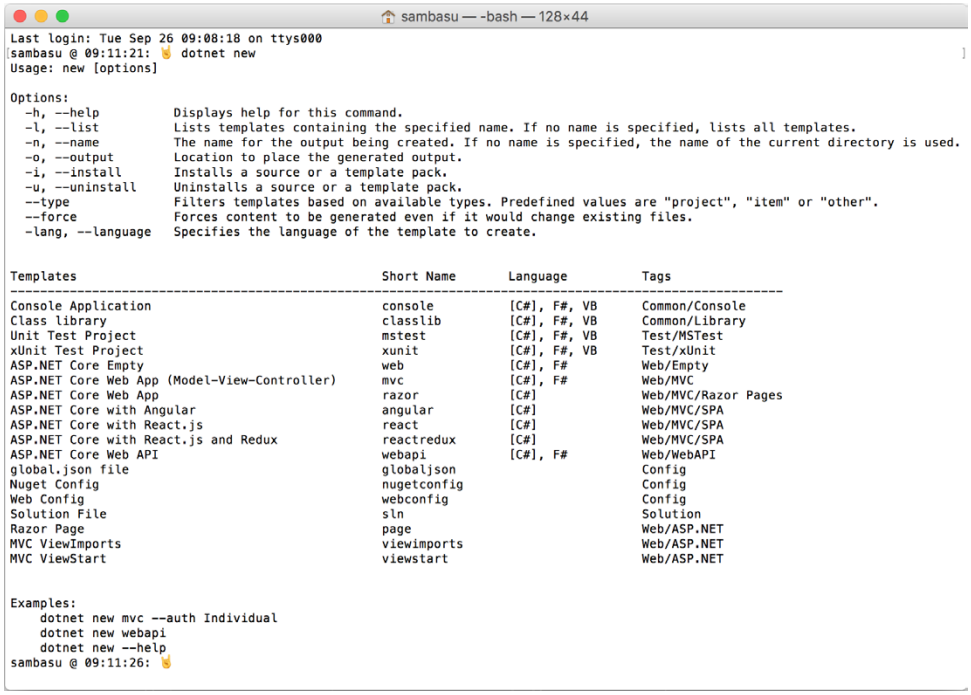
- One of the fastest full-featured Web frameworks Web Framework Benchmarks by TechEmpower
- Full support for .NET Core 2.0 and .NET Standard 2.0
- Backward compatibility to run on .NET Framework 4.6.1
- Combined MVC and Web API stack
- Can act as super-fast API backend for Mobile apps
- New Razor Pages support
- New project templates
- Streamlined support for client-side JavaScript SPA frameworks
- Improved Logging, App Insight and Azure tooling

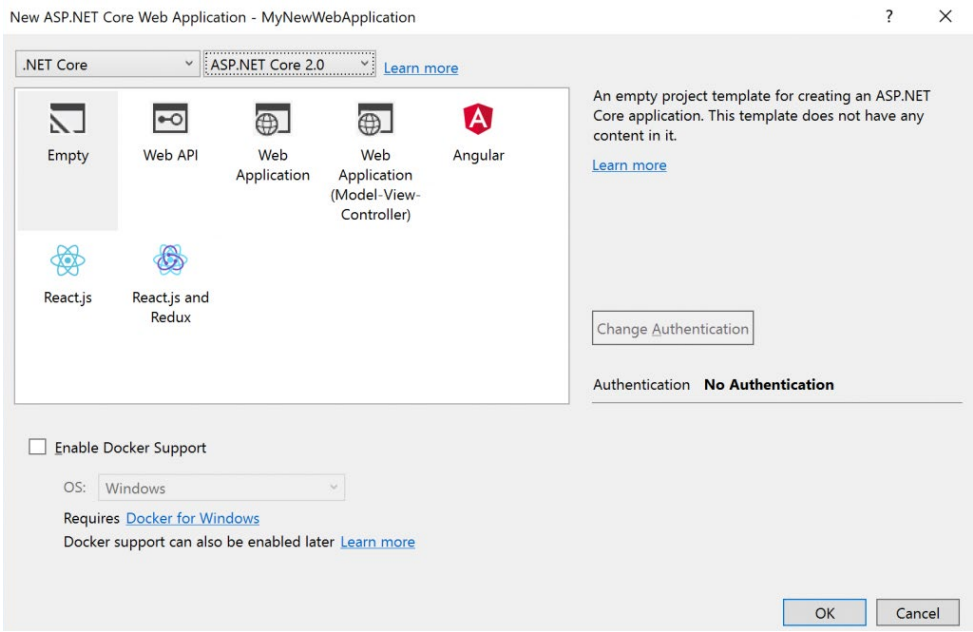
CLI or Visual Tooling

Developers building web apps with ASP.NET Core 2.0 have a plethora of rich tooling to choose from. Here are your choices:

- Visual Studio 2017 (15.3+): a fully-featured IDE, featuring templates and extensions that target ASP.NET Core 2.0
- Visual Studio for Mac: a native macOS IDE with templates for ASP.NET Core 2.0
- Visual Studio Code with the C# extension: a lightweight cross-platform text editor; provides a familiar coding experience and integrated terminal
- Command line tools: truly cross-platform; has all ASP.NET Core 2.0 templates and supports every stage of web app development
- A code text editor with code IntelliSense support through OmniSharp

The good news for developers is .NET tooling is consistent no matter what your development tooling. What CLI can do over commands is exactly what Visual Studio does visually. Look at some of the app templates supported by 'dotnet new' CLI tooling—you'll see corresponding similar templates on doing File | New in Visual Studio. Some of the app templates are what you expect from ASP.NET. Some are unexpected—we'll break things down.





Easy Dependencies

The ASP.NET team has created one of the most streamlined ASP.NET application architectures yet. With ASP.NET Core 2.0, application dependencies are bundled into a single metapackage reference—Microsoft.AspNetCore.All. Each dependency in the metapackage can be referenced individually, however using the bundled approach offers additional benefits. All the features of ASP.NET Core 2.x and Entity Framework Core 2.x are included in the bundled Microsoft.AspNetCore.All package. In addition, applications using the Microsoft.AspNetCore.All metapackage automatically take advantage of the .NET Core Runtime Store. The Runtime Store contains all the runtime assets needed for the application, thus reducing overheads during build/deployments.

```
<ItemGroup>
  <PackageReference Include="Microsoft.
AspNetCore.All" Version="2.0.0" />
</ItemGroup>
```

Easy Portability

One of the nicest things about .NET Core 2.0 and ASP.NET Core 2.0 for developers is portability—the framework and tooling is consistent across all platforms. Developers' choice of development OS doesn't matter—Windows and Mac have an identical experience. Gone is the ASP.NET specific Project.json file for dependencies and configuration.

The familiar .CSProj file is back, as shown below for a boilerplate ASP.NET Core 2.0 project. Whether the ASP.NET project is scaffolded from Visual Studio File | New Project or through the dotnet new CLI tools, the resulting project has the same .CSProj file. It simply points to the target runtime framework and pulls in the .NET dependencies.

This also means that developers can collaborate easier and have ASP.NET projects be portable across machines. It's perfectly conceivable to have multiple developers work on the same ASP.NET project but use different IDEs—one on Visual Studio on Windows, one on Visual Studio for Mac and yet others on simple text editors. The .SLN and .CSProj files are perfectly portable and the code writing experience is the same everywhere.

```
<Project Sdk="Microsoft.NET.Sdk.Web">
  <PropertyGroup>
    <TargetFramework>netcoreapp2.0</
TargetFramework>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="Microsoft.
AspNetCore.All" Version="2.0.0" />
  </ItemGroup>
  <ItemGroup>
    <DotNetCliToolReference
Include="Microsoft.VisualStudio.Web.
CodeGeneration.Tools" Version="2.0.0" />
  </ItemGroup>
</Project>
```

Easy Bootstrapping

ASP.NET Core 2.0 web apps support fast streamlined bootstrapping. This is especially true for setup and initialization routines. The granular details of creating the webhost for the app have been abstracted into the default CreateDefaultBuilder method. Developers retain the flexibility to use the default or create a custom webhost from scratch as needed.

```
public static IWebHost BuildWebHost(string[] args) =>
  WebHost.CreateDefaultBuilder(args)
         .UseStartup<Startup>()
         .Build();
```

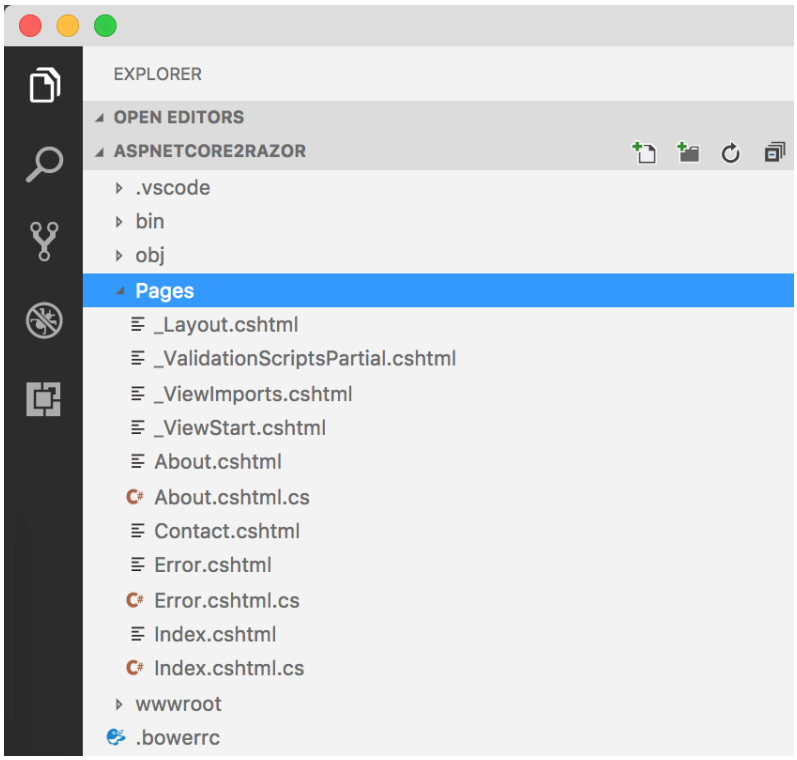
Before ASP.NET Core 2.0, Razor view files were published as .CSHTML files, which were compiled at runtime. This just-in-time compilation came at the expense of a performance hit. In older applications, developers could choose to manually precompile Razor views to reduce the published bundle and increase startup time. Now with ASP.NET Core 2.0 project templates, precompilation is enabled by default thus boosting app startup performance.

Razor Pages

There is a new programming model for simple web pages in ASP.NET Core called Razor Pages. Sometimes, you just want to display some content on a page without going through much ceremony, and maybe do a page postback to server. This is exactly what Razor Pages is meant for. Here are few more pointers about Razor Pages to clear the air of mystery:

- New feature baked into ASP.NET Core MVC
- Auto-enabled for simple page-focused scenarios
- Simple convention-based routing | Razor Pages are served up from Pages directory
- A simple page-level Directive turns Pages into MVC Actions | They handle requests directly
- Razor Pages skip the Controller orchestration | Sort of a simplified Model-View pattern
- Razor Pages are CSHTML Views with optional Code-behind files
- Razor Pages fully support Razor syntax, TagHelpers and MVC validations
- The PageModel contains Handlers to support HTTP verbs
- Razor Pages are not to be confused with WebForms | There is no ViewState between client/server

Here is what ASP.NET Core 2.0 project templates put out for a Razor Pages project. Notice the simplicity of the Pages directory, as compared to a full MVC project:



Let's look at a simple Razor Page view—a CSHTML file with an @page directive on top:

```
@page

=@model AboutModel
@{
  ViewData[“Title”] = “About”;
}

<h2>@ViewData[“Title”]</h2>
<h3>@Model.CodeBehindModelMessage</h3>

<p>Use this area to provide additional information.</p>
```

Notice how the view points to a model and pulls data out of the model. Turns out, the model is defined in a code-behind file with the same name as the view with .CS at the end, simply by convention:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace AspNetCore2Razor.Pages
{
  public class AboutModel : PageModel
  {
    public string CodeBehindModelMessage { get; set; }

    public void OnGet()
    {
      CodeBehindModelMessage = “This is from the code-behind.”;
    }
  }
}
```

Notice how the model implements a PageModel—a base class that binds together much of the Razor Pages features. The page models can define methods that correspond to HTTP verbs like Get, Put, Delete and Post for supporting CRUD operations when the Razor Page posts back to the server. You can see how this simple model helps support putting content out on Razor Pages and adding dynamic features through simple server roundtrips.

Now, here's a little variation to defining a PageModel for a Razor Page, this time in the View itself:

```
@page
@model ContactModel
@using Microsoft.AspNetCore.Mvc.RazorPages

@functions {
  public class ContactModel : PageModel
  {
    public string InPageModelMessage { get; set; }

    public void OnGet()
    {
      InPageModelMessage = "This is from the View.";
    }
  }
}

<h3>@Model.InPageModelMessage</h3>

<address>
  One Microsoft Way<br />
  Redmond, WA 98052-6399<br />
  <abbr title="Phone">P:</abbr>
  425.555.0100
</address>

<address>
  <strong>Support:</strong> <a href="mailto:support@example.com">support
example.com</a><br />
  <strong>Marketing:</strong> <a href="mailto:marketing@example
com">marketing@example.com</a>
</address>
```

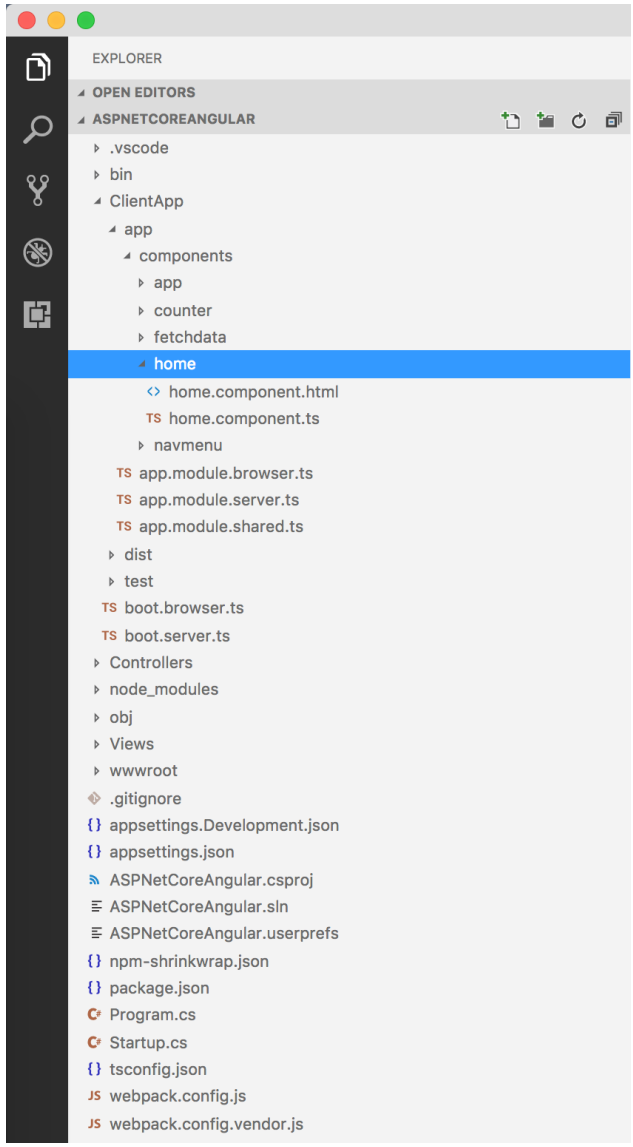
Notice how the PageModel and server-side C# code can be written entirely within a Razor Page view, skipping the code-behind file altogether. While doable, this is not suggested for any complicated pages with substantial code in the PageModel—separation of concerns is always nice.

SPA Templates

Traditional ASP.NET web development mostly relies on server-side rendering with Razor and other view engines. However, browsers have come a long way and JavaScript is ubiquitous. One may argue that client-side web development is where things are headed. ASP.NET Core 2.0 is a modern web framework and in no way attempts to hold developers back in building their web apps as they see fit.

Developers have always been able to bring in JavaScript, CSS and other frontend web development artifacts into the ASP.NET stack. However modern full-featured JavaScript frameworks do a lot out of the box and it doesn't make much sense for developers to reinvent the wheel. There are several popular Single Page Application (SPA) type JavaScript frameworks that aid in building fully frontend web apps with navigation, state management, bundling and the works. Turns out, ASP.NET Core 2.0 works very nicely with these SPA frameworks.

Arguably, two of the most popular JavaScript SPA frameworks are - Angular and React. And ASP.NET Core 2.0 offers built-in templates—both with dotnet CLI or the Visual Studio File | New Project route. If going the Angular route, below is a sample project as scaffolded by the ASP.NET Core 2.0 Angular template:



Developers should notice a few differences right away, compared to traditional ASP.NET MVC projects:

- The Model and View folders aren't there | The Controllers do bare minimum
- The point is to build a true SPA | All the app's functionality is client-side
- Accordingly, the ClientApp folder is where most of the app's code lives
- App features are built as reusable app components
- npm is used to bring in app dependencies
- WebPack is used for configurations and component bundling
- Each component is commonly made up of HTML and CSS | JavaScript pulls it all together
- App business logic is commonly written in TypeScript for Angular 2.0 forward

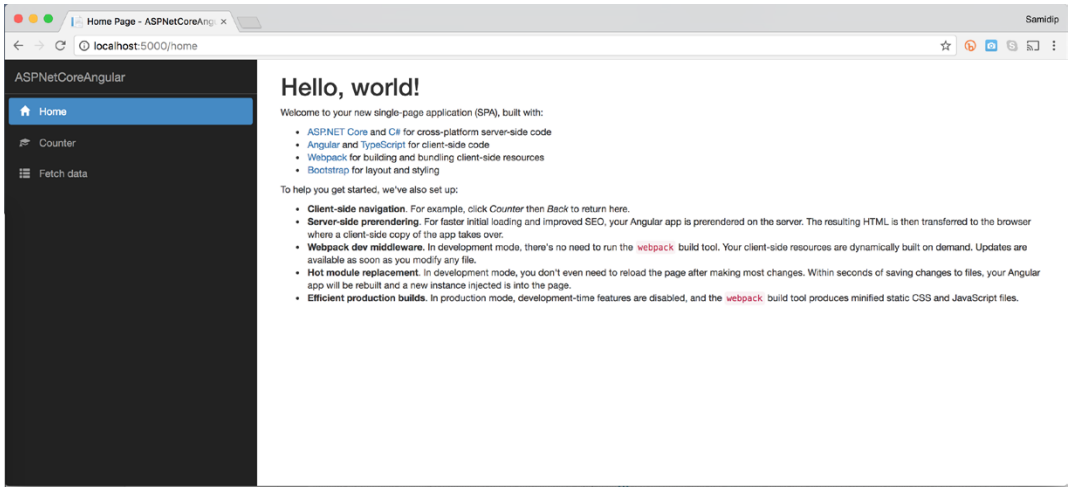
Below is a sample app component in TypeScript:

```
import { Component } from '@angular/core';

@Component({
  selector: 'app',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

export class AppComponent { }
```

Once the ASP.NET Core 2.0 Angular template is done scaffolding, developers have a familiar dotnet run or F5 experience to run the app. Notice the app template being different from traditional ASP.NET apps. The experience when using the ASP.NET Core 2.0 React template is identical.



The Angular/React templates in ASP.NET Core 2.0 are meant to get .NET developers to a good starting point with SPA projects. They also include abstractions to fast-track newcomers. Like any other project templates, these new SPA templates aren't required, but rather helpful. Development teams can always build two separate projects—a fully frontend SPA web app with JavaScript frameworks and an ASP.NET WebAPI project serving up data from the backend. ASP.NET is quite happy in this role and will support APIs for a variety of client apps. But most modern SPA/JavaScript web apps need a lot of orchestration—configuring services, bundlers and managing dependencies. This "wild west" approach may be intimidating for many .NET developers and this is where the ASP.NET Core 2.0 Angular/React templates come in—they have pre-configured frontend tooling and hook up a bunch of things for developers behind the scenes.

Here's a handful of benefits that come with the SPA templates in ASP.NET Core 2.0:

- Support for server-side pre-rendering of JavaScript components | Embeds NodeJS in ASP.NET runtime hosting
- Webpack Hot Module Replacement | Speeds up dev/test cycles by automatically refreshing components on TypeScript/JavaScript/CSS edits
- Node Module dependency management | All of what Angular/React need are referenced and managed
- Webpack integration is built-in | Does bundling, minification and TypeScript compilation

Telerik and Kendo UI for Web

As you can see, modern ASP.NET comes in variety of flavors—developers can mix/match server side C# with client-side JavaScript. One thing stays true though—web apps need polished UI to stand out and developers should not reinvent the wheel. Progress offers various UI suites for lighting up your web apps, no matter how you build them:

- Telerik UI for ASP.NET AJAX - for WebForms apps
- Telerik UI for ASP.NET MVC - for traditional MVC web apps
- Telerik UI for ASP.NET Core - support for ASP.NET Core and TagHelpers
- Kendo UI for jQuery - UI widgets for jQuery-based web apps

You can download free trials of all of them through the unified installer of the **DevCraft bundle**

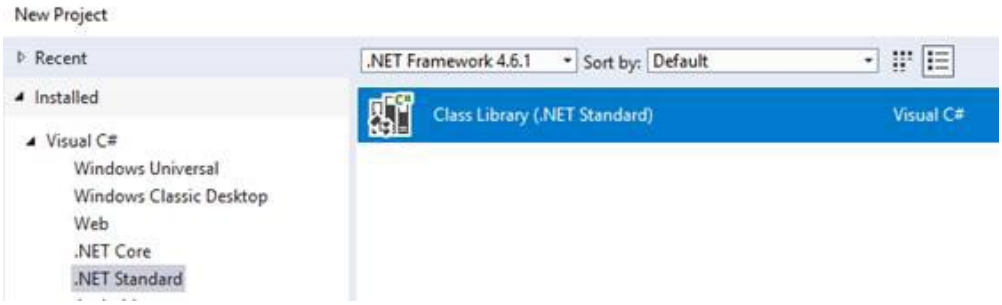
And if you're building your UI with JavaScript, take a look at:

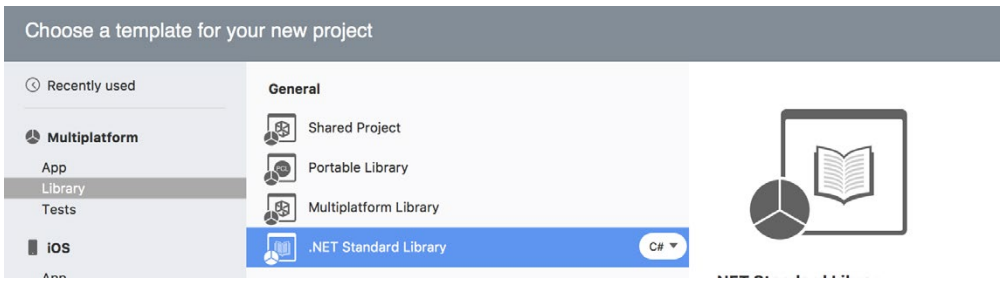
- Kendo UI with support for Angular - UI components for Angular-based web apps
- Kendo UI with support for React - UI components for React-based web apps
- Kendo UI with support for Vue - UI components for Vue-based web apps

Better Mobile Apps with Xamarin

Congratulations! You have picked Xamarin to build your next cross-platform mobile app. You get to reuse your existing skills in C#/XAML and build a truly native cross-platform app from a single code base. For .NET developers, Xamarin has almost single-handedly democratized cross-platform development—you can target apps running on just about every platform and device.

All Xamarin apps—Xamarin.iOS/Android and Xamarin.Forms—work with .NET Standard libraries. Xamarin developers should look to abstract out any code that be shared and this is best done as a .NET Standard library for maximum portability. The dream of re-using code between Xamarin and other .NET apps is a reality, thanks to the common API platform offered by .NET Standard. .NET Standard libraries can be created from scratch in Visual Studio, Visual Studio for Mac or CLI:





Any .NET Standard library can be consumed inside Xamarin apps, whether it's directly from NuGet or through third party sources. Developers can use .NET Standard dependencies in their Xamarin apps with no changes to existing dependencies required. In fact, the recent Xamarin.Forms 2.4.0 stable release has full support for .NET Standard 2.0. Good times ahead for Xamarin developers trying to share code across platforms.

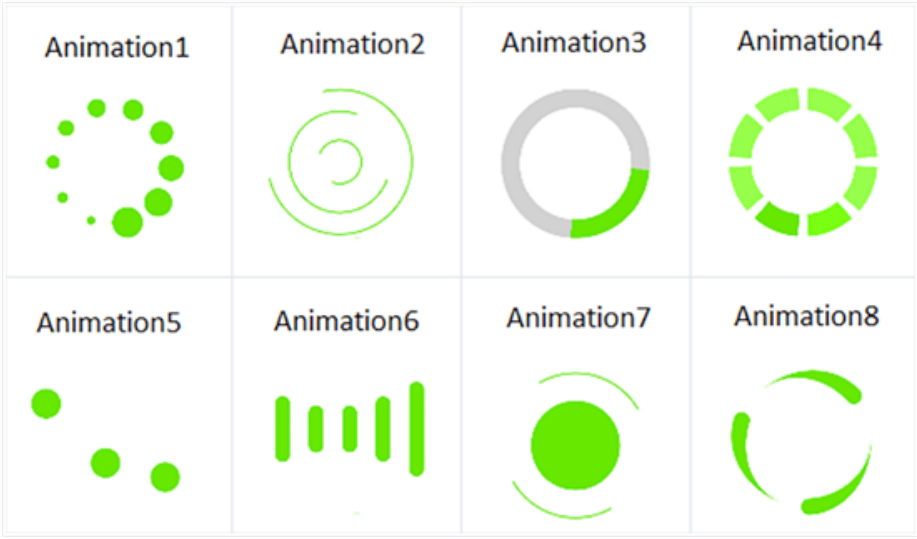
Telerik UI for Xamarin

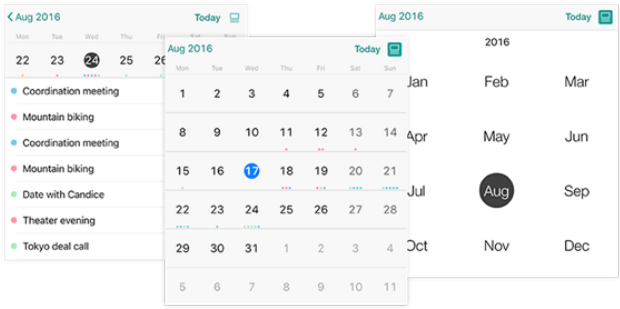
You are completely sold on the Xamarin promise. You have grabbed the IDE/tools of your choice, started building your dream Xamarin app and incorporated the necessary frameworks. Guess what? Your end users do not see any of magic behind the scenes. What they do see is the app UI and the fluidity of user experience that your app provides. You can try reinventing the wheel on complex UI or go for some well-engineered UI controls out of the box. Sure, there are a few offerings, but let's talk about what is arguably the most polished and developer-friendly.

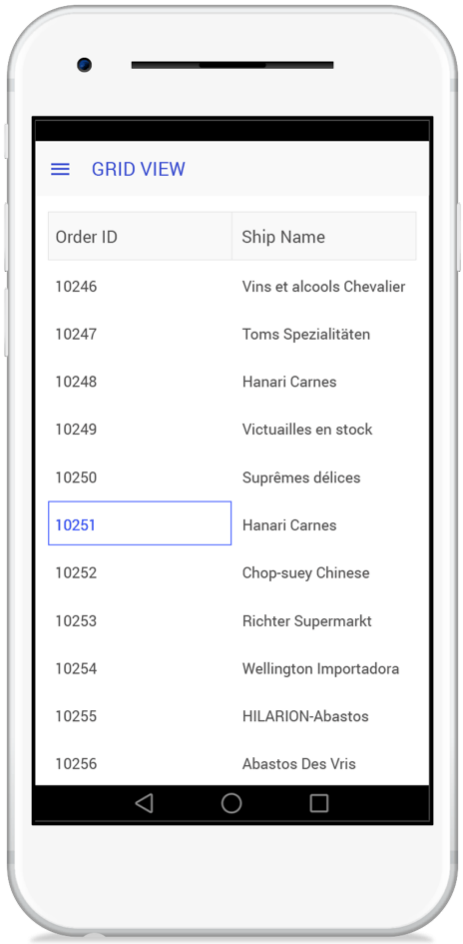
Telerik UI for Xamarin includes elegant, polished and performant native UI widgets for all your Xamarin apps. With UI for Xamarin, you'll get to deliver your Xamarin apps quicker and delight users with beautiful functional UI.

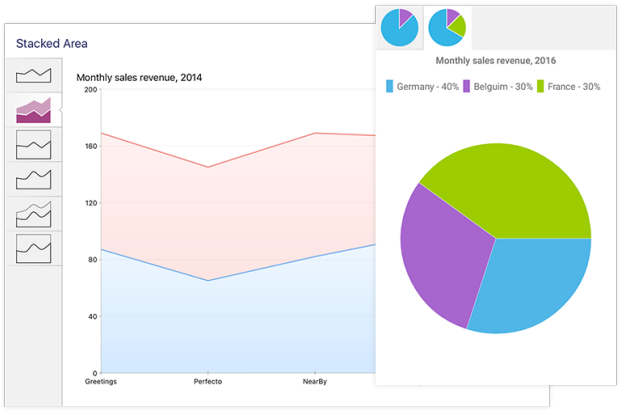
It is easy to integrate Telerik UI for Xamarin in your existing projects—get the NuGet package for Visual Studio on Windows/Mac or just download the bits. If you have the luxury of starting greenfield projects, Telerik UI for Xamarin provides some simple Visual Studio Templates to get you to a good starting point—again both on Windows or Mac.

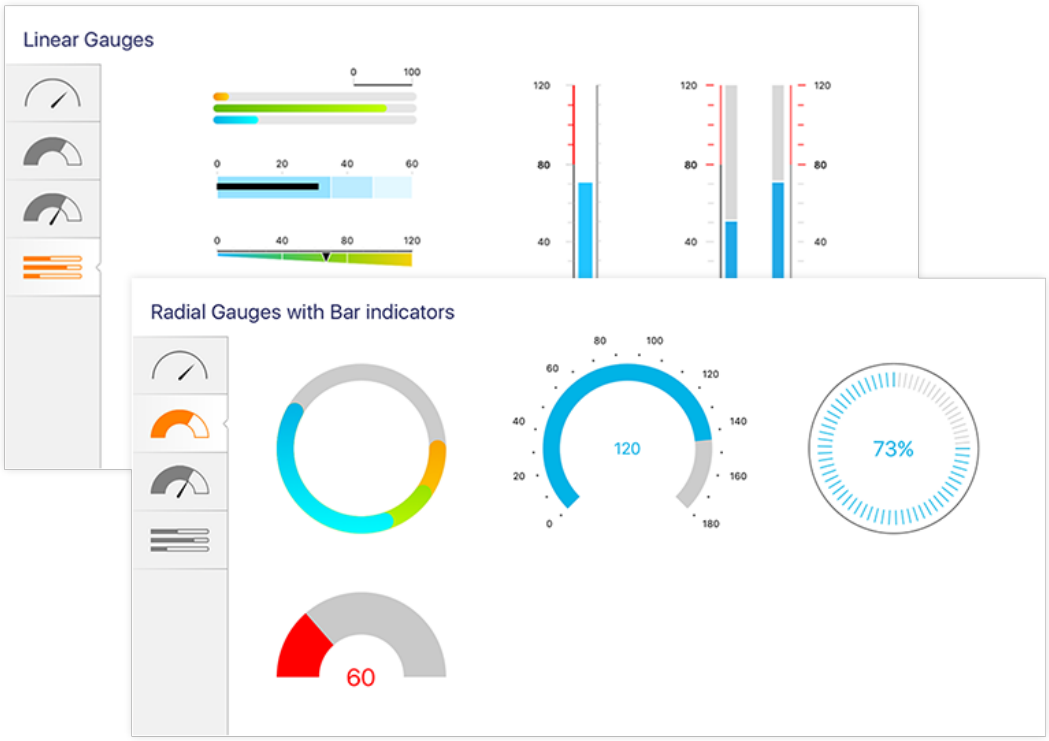
What do you get with UI for Xamarin? Beautiful, professionally engineered and highly performant UI to delight users. You have gorgeous Charts, must-have ListViews, LOB-friendly DataGrids, handy Gauges, pixel-perfect Calendars, magical SideDrawers, super-helpful DataForm, essential TabView and much more. Powering Telerik UI for Xamarin is truly native UI for each platform, with support for Xamarin.iOS, Xamarin.Android and Xamarin.Forms.

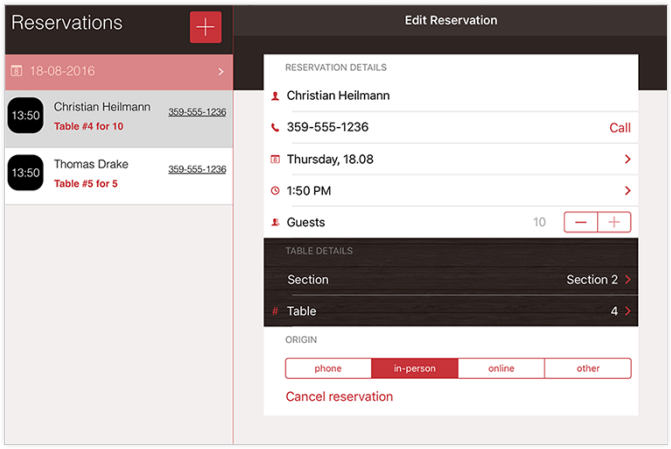


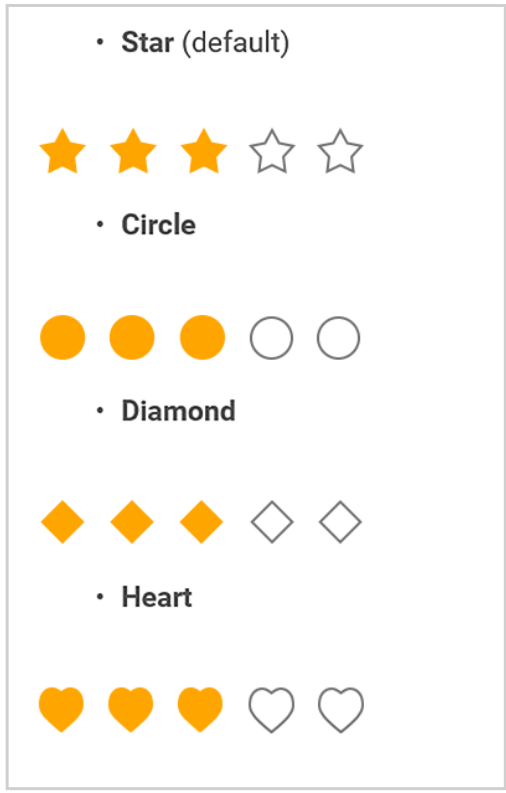


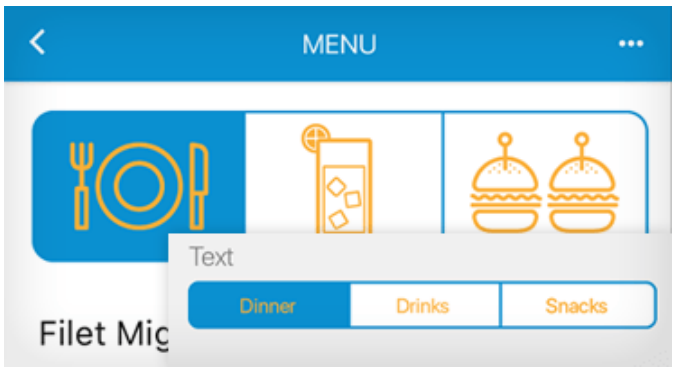


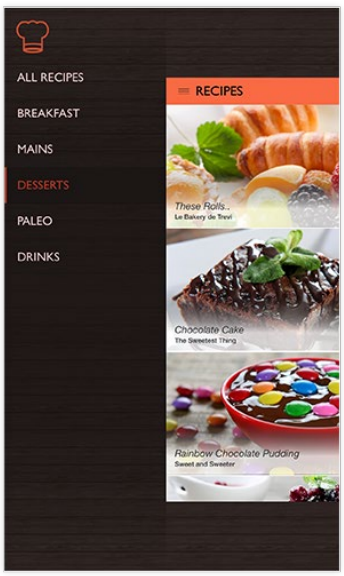


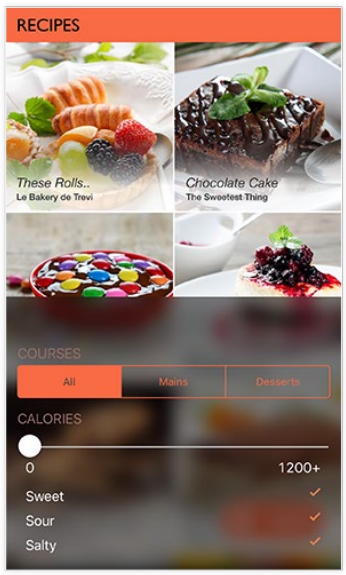


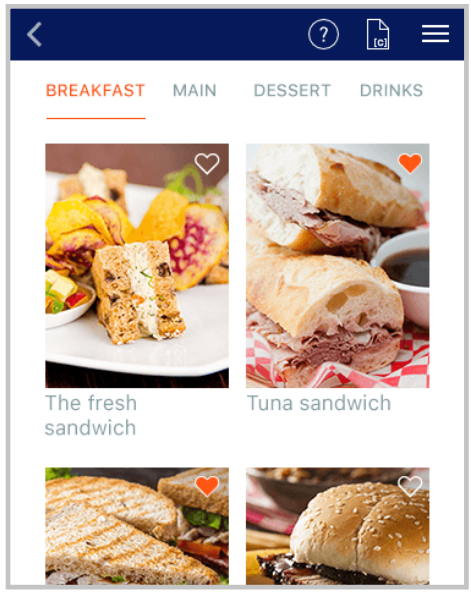










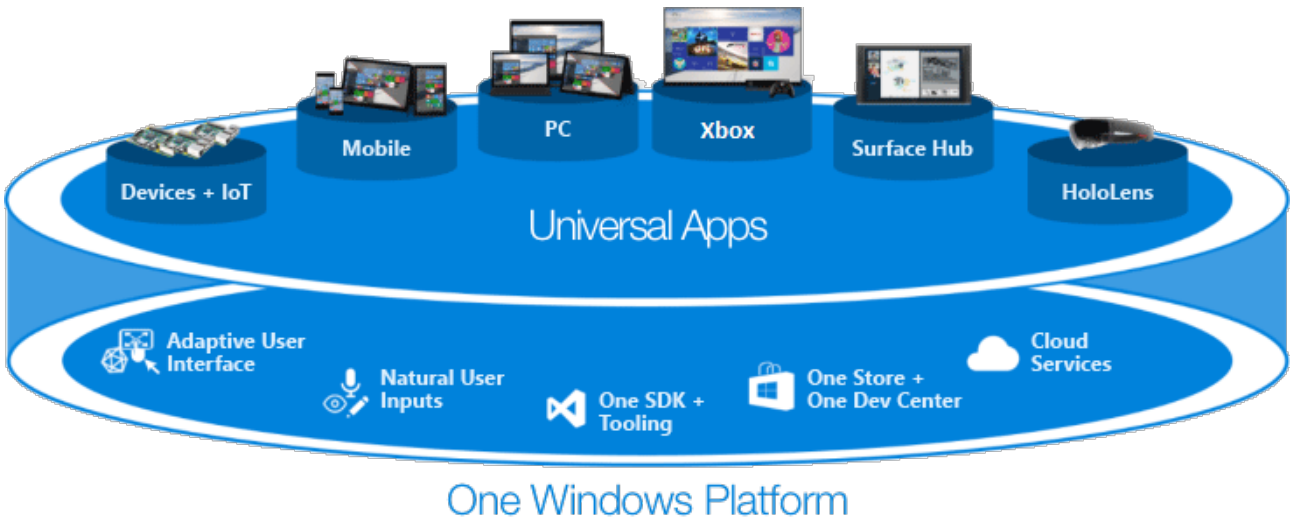


Better Windows Apps with UWP

Over the past decade, Windows has undergone many changes to support industry trends and reflect Microsoft's vision for its flagship operating system. Numerous features and improvements were added over the years with Windows 7 and Windows 8. However, it wasn't until Windows 10 that things became interesting.

Windows 10 represents a significant milestone for Windows as a platform for both users and developers. Unlike previous versions, Windows 10 isn't confined to the desktop. Instead, it runs across a wide range of devices and form factors. This includes IoT devices (running on ARM or x86/x64 architectures) and platforms like the Xbox, HoloLens, and Surface Hub. For developers, Windows 10 also introduces a new way for creating apps called the Universal Windows Platform (UWP).

UWP represents an exciting opportunity as it allows developers to target a rich ecosystem of Windows-based devices through a unified API. It achieves a long-standing goal at Microsoft of having one—and only one—version of Windows to target. Through this unified API, developers can now build applications that can run across devices.



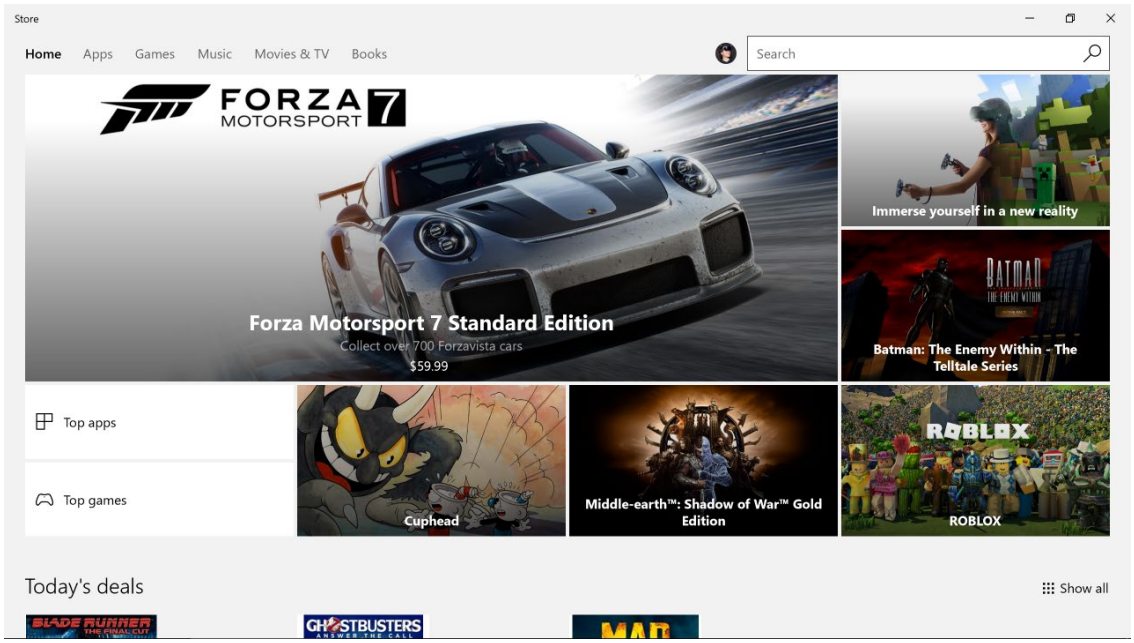
Source: Intro to the Universal Windows Platform

Not all devices are the same. Some support capabilities that are unique. UWP supports the ability to target these capabilities through Extension SDKs. These device-specific APIs can be targeted by developers to take advantage of these features.

In addition to leveraging device-specific APIs, UWP also enables developers to build applications with adaptive controls and input. This is a concept known as Responsive Design that's being populated in web development circles. This is the ability for applications to work across devices with differing screen dimensions and forms of user input (i.e. Xbox One controllers).

Windows Store

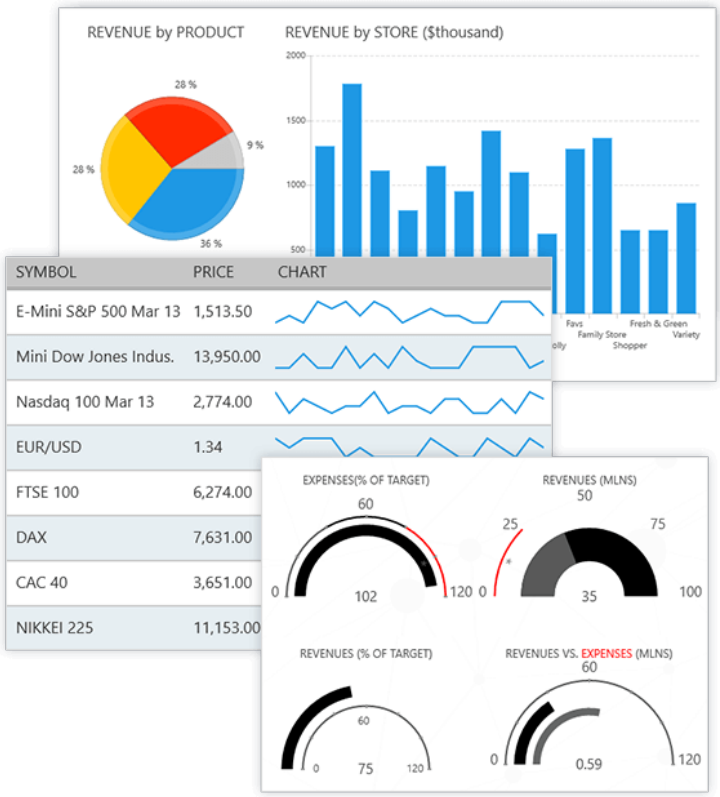
One of the biggest changes introduced with Windows 8 and featured prominently in Windows 10 is the Windows Store. This is Microsoft's distribution platform for free and commercial apps. It's a curated environment that requires developers to have their applications certified prior to publication. This may seem like a significant hurdle, but one that's worthwhile when you consider that it makes these applications available to millions of Windows users.



The Windows Store uses the .AppX packaging format, which is a step up from the traditional model of executables and libraries. The .AppX packaging format allows developers to express their application's intent. This enables them to express which capabilities they require for applications to operate correctly on Windows 10. This way, users can download applications from the Windows Store with a higher level of trust over the traditional application model.

Telerik UI for UWP

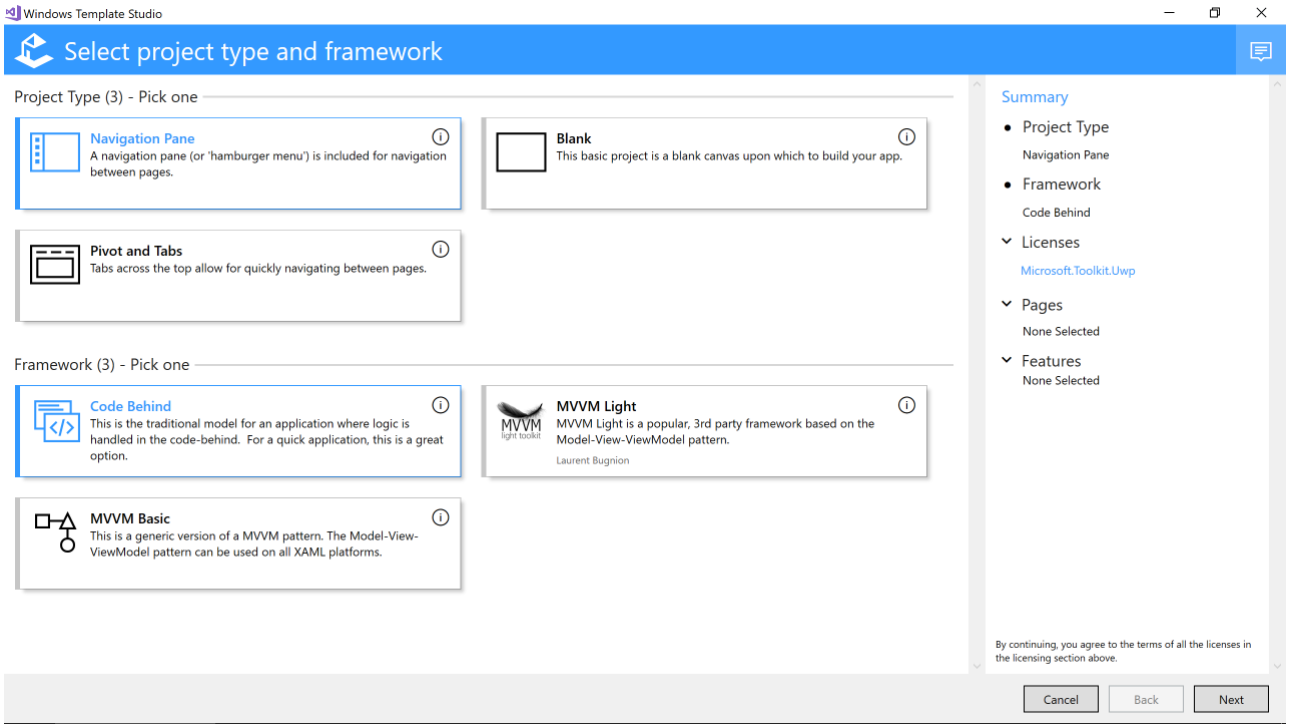
Telerik UI for UWP is a suite of 20+ UI controls that help developers build UWP applications. It contains controls that address common UI requirements in line-of-business (LOB) applications, including data management (DataForm), scheduling (Calendar), navigation (RadialMenu), data visualization (Chart), and more. The source code for these controls can be found on GitHub: UI-for-UWP.



This controls can be easily incorporated into existing UWP projects via NuGet. Once added, controls can be added and qualified in XAML pages through prefixes and namespaces. Here's an example using the RadialGauge control:

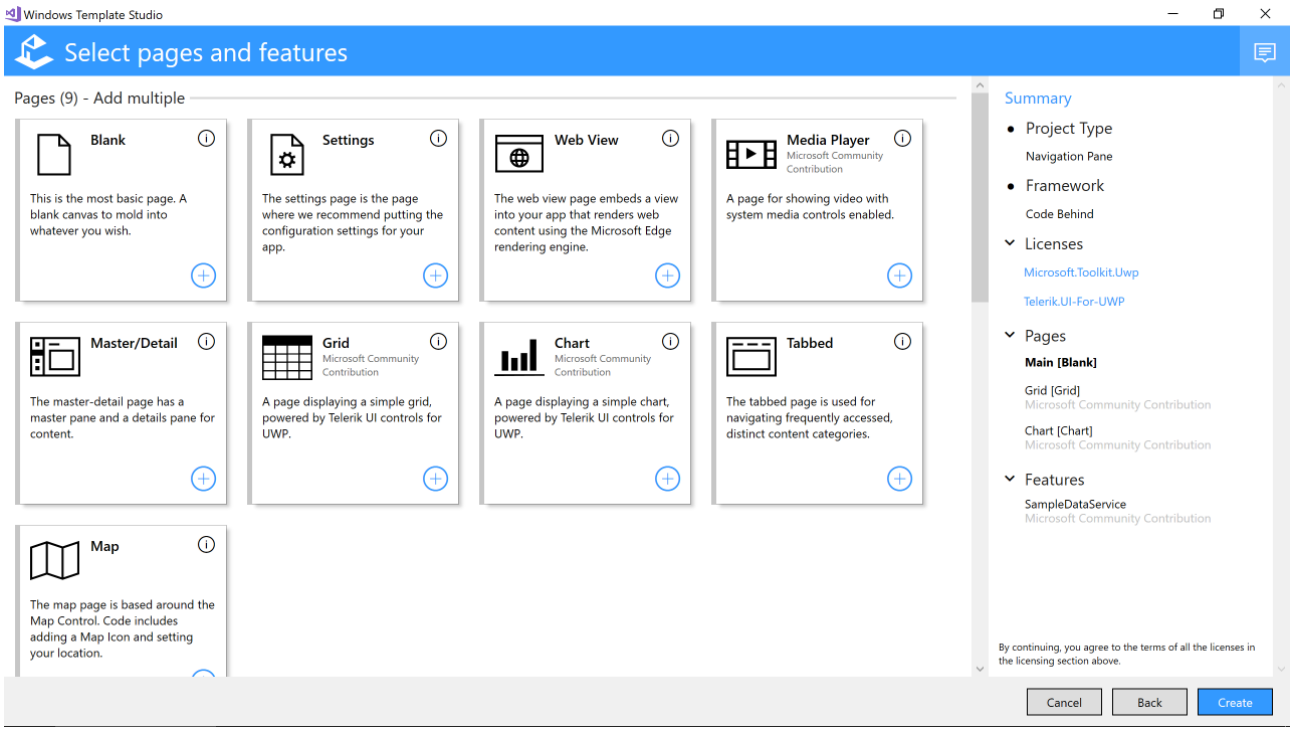
```
<Page xmlns:telerik=”using:Telerik.UI.Xaml.Controls.DataVisualization”>
  ...
  <telerik:RadRadialGauge>
    ...
  </telerik:RadRadialGauge>
</Page>
```

These controls can be used with controls from other libraries such as the UWP Community Toolkit. This combination empowers developers to build extremely powerful and versatile UWP applications. Incorporating these controls into existing applications is fairly trivial. That stated, if developers looking for a guided approach towards building new applications with Telerik UI for UWP should check out Windows Template Studio:

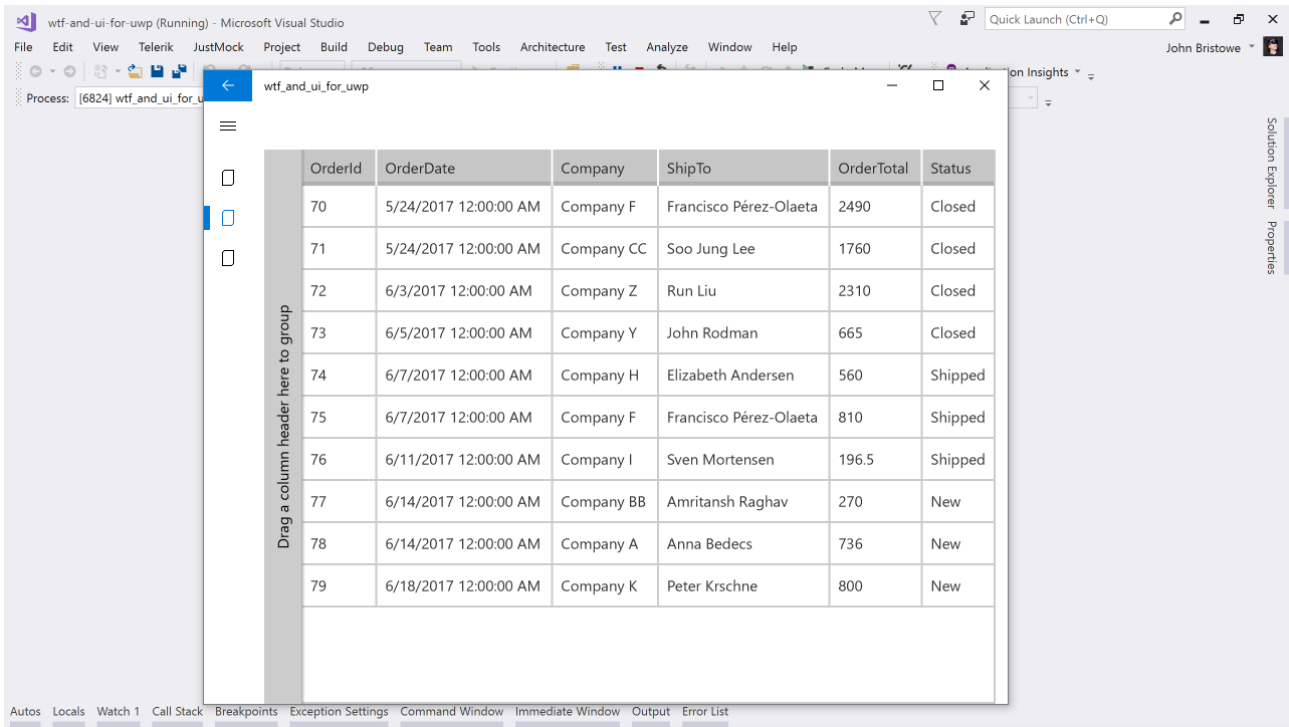


This extension for Visual Studio will generate a UWP application through templates. The goal is to get you up and running quickly with a project structure and source files that can be modified afterward.

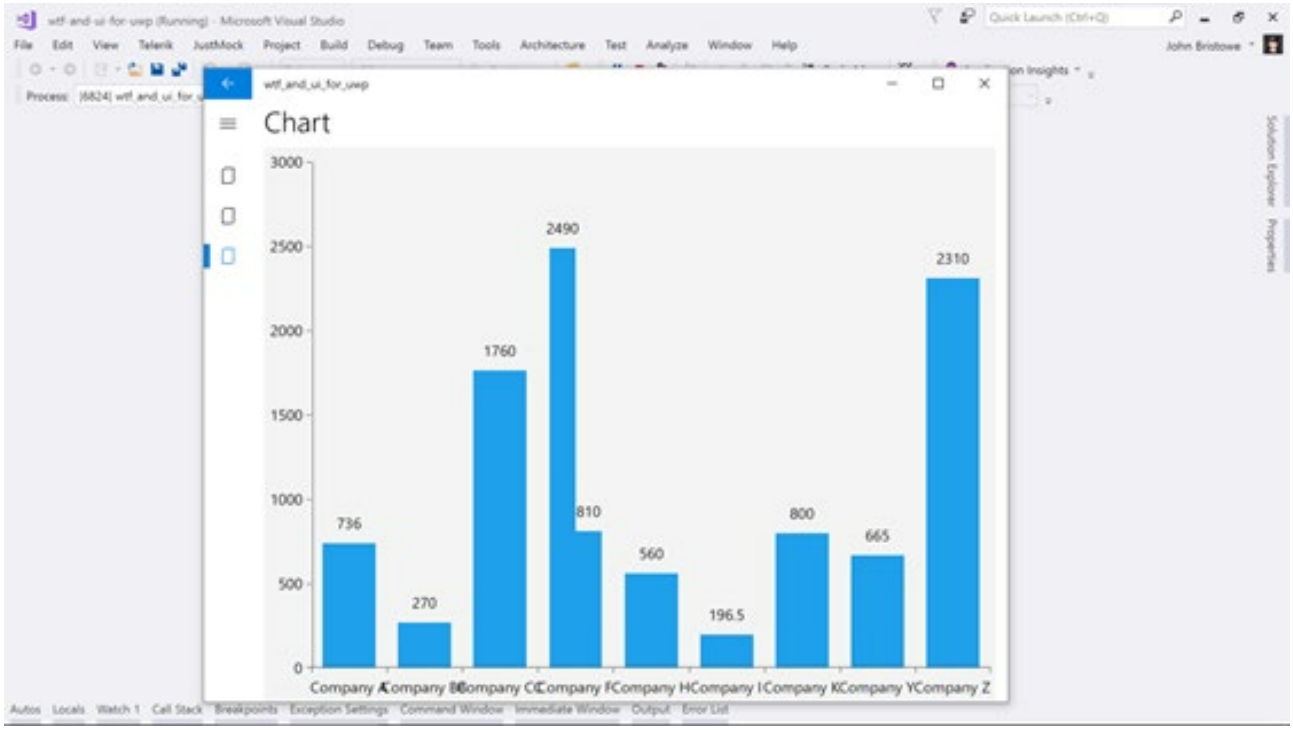
Recently, Windows Template Studio added templates for the Chart and Grid controls from Telerik UI for UWP:



Selecting either of those controls will generate pages will the necessary XAML and code needed to display them:



Windows Template Studio is available to download and install from the Visual Studio Marketplace.



Conclusion

As you've seen in this whitepaper, .NET is well-positioned for upcoming success. Its new capabilities—including fundamental improvements and the standardization efforts surrounding them—will help incubate the best ideas for modern .NET development.

Authors

John Bristowe

Principal Developer Advocate with Progress, living in Australia. Prior to joining Progress, he was the Senior Developer Evangelist with Microsoft.

Ed Charbeneau

A web enthusiast, speaker, writer and Developer Advocate with Progress. He has designed and developed web-based applications for business, manufacturing, systems integration as well as customer-facing websites. Ed enjoys geeking out to cool new tech, brainstorming about future technology and admiring great design.

Sam Basu

Technologist, author, speaker, Microsoft MVP, gadget-lover and Progress Developer Advocate. With a long developer background, he now spends much of his time advocating modern web/mobile/cloud development platforms on Microsoft/Telerik technology stacks. His spare times call for travel, fast cars, cricket and culinary adventures with the family.

This resource is brought to you by DevCraft.

This convenient bundle includes a wide-range of UI, reporting and productivity tools for both .NET and JavaScript technologies and support that's got your back in every step of your project.

Thanks to our intuitive APIs, alongside thousands of demos with source code availability, comprehensive documentation and a full assortment of VS templates, you will get up and running with our tools in no time and fully embrace your inner ninja.

By leveraging the broad array of themes, skins, styling and customization options, your application will awe even the best front-end designers.

Learn more

About Progress

Progress (NASDAQ: PRGS) offers the leading platform for developing and deploying mission-critical business applications. Progress empowers enterprises and ISVs to build and deliver cognitive-first applications that harness big data to derive business insights and competitive advantage. Progress offers leading technologies for easily building powerful user interfaces across any type of device, a reliable, scalable and secure backend platform to deploy modern applications, leading data connectivity to all sources, and award-winning predictive analytics that brings the power of machine learning to any organization. Over 1,700 independent software vendors, 100,000 enterprise customers and 2 million developers rely on Progress to power their applications. Learn about Progress at www.progress.com or +1-800-477-6473.

Worldwide Headquarters

Progress, 14 Oak Park, Bedford, MA 01730 USA
Tel: +1 781 280-4000 Fax: +1 781 280-4095
On the Web at: www.progress.com
Find us on facebook.com/progresssw twitter.com/progresssw youtube.com/progresssw
For regional international office locations and contact information, please go to www.progress.com/worldwide

Rev 2017/11 | 171102-0094

Progress