Geolocation, Vue, .NET Tooling, ExpressJS

CODE

NOV
DEC
2020

codemag.com - THE LEADING INDEPENDENT DEVELOPER MAGAZINE - US $ 8.95 Can $ 11.95

Ready, Set, Building Applications with GO!

Using the .NET Core Tools Framework

Building Geolocation-Enabled Applications

Getting Started with ExpressJS and TypeScript

You're invited *to join Scott Guthrie on December 9 and Charles Lamanna on December 10 as they chat with former CNN anchor, George Howell*

Register today – complimentary

12 PM East • 9 AM Pacific

DEC 9 *A Fireside Chat on trending topics in today's world,* with **Scott Guthrie**, Executive VP, Microsoft

DEC 10 *Accelerating Developers with the Microsoft Power Platform and live Q&A* with **Charles Lamanna**, Corporate VP, Microsoft

Keynotes hosted by **George Howell**, former CNN Anchor and international journalist

Following the chats, join us at the

DEVintersection *Virtual Workshop* Event DEC 9+10, 2020

4-HOUR WORKSHOPS *only* $199 each 1–5 PM East • 10 AM–2 PM Pacific

Dec 9 Workshops

Advanced .NET, Containers and Azure
Scott Hunter, Brady Gaster, Paul Yuknewicz, Sourabh Shirhatti, and Maria Naggaga Nakanwagi

Azure Synapse Analytics – The Ultimate Data Engineering and Data Science Platform – Day 1
Ciprian Jichici, Carey Payette

Become a Better C# Programmer – Leverage Newer Features
Kathleen Dollard

Building Blazor Applications – Day 1
Carl Franklin

Deep dive into Azure Security – Day 1
Chris Givens

Domain Driven Design Patterns
Steve Smith

Modernize Your ASP.NET Web Forms Application to .NET Core, Blazor, and the Cloud
Jeff Fritz

Dec 10 Workshops

Power Platform for Pro Devs
Per Mikkelsen, Ryan Cunningham

Azure Synapse Analytics – The Ultimate Data Engineering and Data Science Platform – Day 2
Ciprian Jichici, Carey Payette

Building ASP.NET Core Apps with Clean Architecture
Steve Smith

Building Blazor Applications – Day 2
Carl Franklin

Deep dive into Azure Security – Day 2
Chris Givens

Get Your Enterprise Microservices Game on in Azure
Michele Leroux Bustamante, Jim Counts

Getting Started as a Data Engineer
Tim Chapman

Sponsors: SentryOne™ Progress®

virtual.devintersection.com **info@DEVintersection.com (203) 527-4160 M-F, 12–4 Eastern**

DEVintersection
SQLintersection
POWERED BY
Microsoft & DEVintersection

<anglebrackets/>

Microsoft Azure
+AI Conference
CO-PRODUCED BY
Microsoft & DEVintersection

June 8–10, 2021
Workshops June 6, 7, 11
Orlando, FL
WALT DISNEY WORLD SWAN

Dec 7–9, 2021
Workshops December 5, 6, 10
Las Vegas, NV
MGM GRAND

Powered by
Microsoft
NextGen
.NET Rocks!
solliance
SQLskills

SCOTT GUTHRIE
Executive Vice President, Cloud + AI Platform, Microsoft

SCOTT HANSELMAN
Principal Program Manager, Web Platform, Microsoft

JULIA LIUSON
Corporate Vice President, Microsoft

CHARLES LAMANNA
Corporate Vice President, Microsoft

DONOVAN BROWN
Principal DevOps Manager, Microsoft

KATHLEEN DOLLARD
Principal Program Manager, Microsoft

If your passion is technology, take it to the next level!

Get a unique perspective from industry experts out in the trenches
Have direct interaction with your favorite Microsoft VPs and execs giving you the insider scoop on what's coming
Find the top speakers sharing real-world solutions and techniques to sharpen your skills for instant ROI

Microsoft and industry experts
Full-day workshops Evening events

@DEVintersection
@AzureAIConf

DEVintersection.com **203-527-4160** M-F, 12pm-4pm EDT **AzureAIConf.com**

Features

8 **A Simple ExpressJS and TypeScript Project**
Sahil walks you through a simple ExpressJS-based Web app using TypeScript to help you get started using these great technologies together.
Sahil Malik

14 **Using Geolocation and Google Maps**
You want people to be able to find your business, or your house, or some meeting place without leaving your website to find it on a map. Using geolocation, the search can be personalized, and Paul shows you that it's easier than you'd think.
Paul Sheriff

24 **Vue's Composition API**
There's no sleight of hand here, as Shawn reveals the magic of Vue's Composition APIs.
Shawn Wildermuth

30 **Introduction to the Go Programming Language**
Although it's been around for a while, Go is coming into its own as companies like Google, Dropbox, and Uber adopt it. Wei-Meng shows you why they're rushing to use Go and why you should too.
Wei-Meng Lee

42 **Using .NET Core Tools to Create Reusable and Shareable Tools and Apps**
The new .NET Core Tools are a simple way to create, publish, and consume an existing NuGet infrastructure and share the tools you build. Rick shows you how.
Rick Strahl

59 **Blockchain: A Practical Application**
John's intrigued by the potential of blockchain, and he takes you along as he explores this interesting tool.
John Petersen

65 **The Complete Guide to Suspense in Vue3**
Suspense is a tool that makes sure your content is ready before displaying it. Bilal explains this complex concept using example app code.
Bilal Haidar

Columns

54 **Talk to an RD: Dr. Neil Roodyn and Markus Egger**
Markus and Neil continue their conversation about the effects of the COVID-19 pandemic on corporate security.
Markus Egger

74 **CODA: What Makes a Leader: An Objective Analysis**
John uses the dictionary to describe what makes a good leader more than just the boss.
John V. Petersen

Departments

6 Editorial

19 Advertisers Index

73 Code Compilers

US subscriptions are US $29.99 for one year. Subscriptions outside the US pay $49.99 USD. Payments should be made in US dollars drawn on a US bank. American Express, MasterCard, Visa, and Discover credit cards are accepted. Bill Me option is available only for US subscriptions. Back issues are available. For subscription information, send e-mail to subscriptions@codemag.com or contact Customer Service at 832-717-4445 ext. 9.

Subscribe online at www.codemag.com

CODE Component Developer Magazine (ISSN # 1547-5166) is published bimonthly by EPS Software Corporation, 6605 Cypresswood Drive, Suite 425, Spring, TX 77379 U.S.A. POSTMASTER: Send address changes to CODE Component Developer Magazine, 6605 Cypresswood Drive, Suite 425, Spring, TX 77379 U.S.A.

LEADTOOLS v21

DOCUMENT ANALYZER SDK

Recognize Thousands of Documents Like Never Before

The LEADTOOLS Document Analyzer SDK Library intelligently identifies document components and zones in text-based office documents (DOC, DOCX, XLS, XLX), PDFs, and document images (JPG, TIFF, PNG PDF) for automated document processing and smart data extraction. The Document Analyzer automatically finds key phrases within structured and unstructured documents, even if the layouts between files are completely different. Additionally, the component performs deep analysis to further improve detection ensuring that nothing is missed and all data of interest is found.

Get Started Today

DOWNLOAD OUR FREE EVALUATION

LEADTOOLS.COM

Microsoft .NET iOS macOS

The New Normal

Let me fill you in on a little secret: Sometimes the editorials you see here just seem to flow from my brain, through my fingers, and onto the page. Other times, it's a real slog just coming up with the general concept of the editorial, let alone the content. When it came to the editorial for this issue, it was the latter rather than the former. It was a slog. The question came up: How do I get myself mentally out of the mud and onto a dry road toward an editorial?

My general strategy is to seek out material that helps me find inspiration either for the concept or to add material to my already existing theme. In this case, I was looking for inspiration to support my already existing theme which, by the title, you can see is: The New Normal.

I've been sitting on this idea for the last few months and was trying to find the proper words to start. This is where I got stuck. Generally, I start my editorials "old-school" by writing them on paper. If you look at **Figure 1**, you'll see that the start of the editorial had a bunch of crossed out lines. I was way stuck.

I pulled myself out of the morass by looking for inspiration. I decided to look at what I wrote for the Nov/Dec 2019 issue. The Nov/Dec editorial was an interesting one as I wasn't the author of that editorial. That editorial was written by my editor, Melanie Spiller, and is called, "Sock, Sock, Shoe, Shoe" (https://codemag.com/Article/1911011/Sock-Sock-Shoe-Shoe). I loved this editorial, as Melanie wrote about her experience of taking her normal morning walk but being forced to reverse the route as there was construction on her normal route. What Melanie learned on this walk was there was a whole universe of things she never noticed even though she had walked the same route for years. Melanie's perspective had shifted.

This leads me to what I want to convey in this editorial, a discussion of the "New Normal." Like it or not, we're faced with a lot of change, some of which may be permanent. For many of us, our world has been turned upside down and we've been forced to take a new route. This change in our world view isn't because of any man-made activity this time, but because of the spread of the COVID-19 virus sweeping the globe. There are so many changes to our way of life! The primary change is, of course, that we must now limit contact with other humans. This is probably the most difficult of all the changes to our world. I'm a social being and I bet that many of you are as well. This limitation has caused a massive ripple effect to the entire world's workforce and that's especially true of software developers.

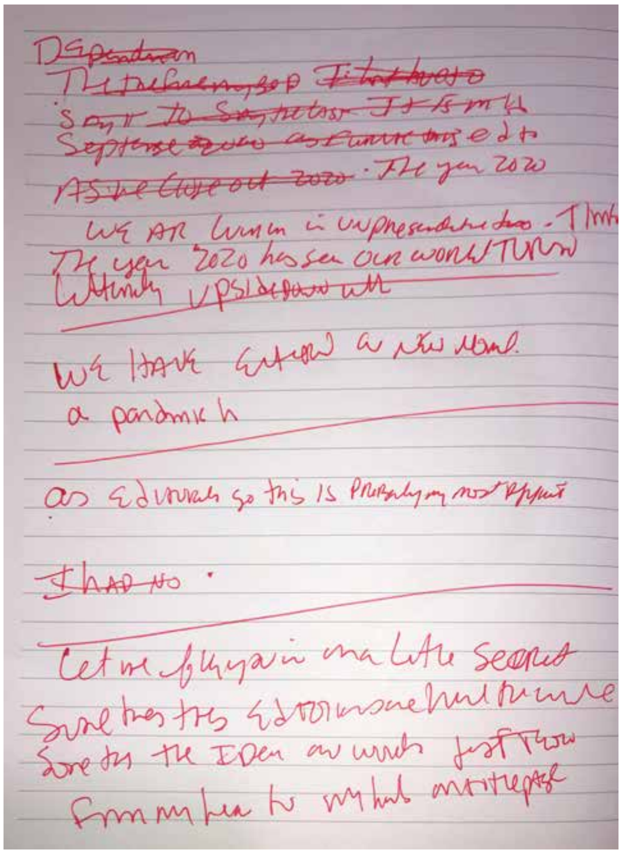
As a software developer, I find myself on planes numerous times a year. Over the span of 25 years, I've flown 2,000,000 miles to attend user groups, meet with clients, or attend/speak at technical conferences. For me, this component being completely absent is probably the biggest adjustment. It's mid-September as I write this, and it's been a full six months since my last flight. That's a new record for me.

That leads me to the next part of the new normal: virtual technical conferences. The concept of in-person conferences is out of the question so we must have our conferences online. This's a big shift because it limits one of the best features of conferences: the hallway talk. Although it's easy to give a technical talk online, it's very difficult to replicate the hallway aspect of in-person conferences. How are you handling the social aspects of your technical regimen?

Another consideration is working from home. For me, this wasn't a huge adjustment as I've been working at home in one form or another for two decades. The biggest adjustment for me is that I now have a high-school-aged child home taking online virtual classes. I consider myself lucky not to have an elementary-school-aged child at home.

Those two areas are just the tip of MY iceberg. I appreciate the fact that we all have our own unique challenges difficulties to deal with. My hope is that once we have better treatments for COVID-19 and a vaccine to prevent its spread, we can take away the good parts of this new normal and build on them. For instance, if you've adapted to the lifestyle of working at home, you may want to re-evaluate where you live. Your choice of where to live might now be limitless. Or if you want to attend a conference, you may find that some of them continue to offer online versions and that you prefer that for a multitude of reasons. I may attend some at home and some in-person, once I have the choice.

I'm hopeful that we'll return to some semblance of normal in the coming months. It's yet to be determined what the definition of normal will be in the coming years but I, for one, am optimistic that we'll end up in a better place than before. That better place will partially be the result of taking a new and unexpected direction, much like Melanie talked about in her 2019 editorial.



~~[illegible]~~
~~[illegible]~~
~~[illegible]~~
~~September 2020 as [illegible]~~ edit
~~As we close out 2020~~. The year 2020
We are living in unprecedented times. Think
The year 2020 has seen our world turned
~~Literally upside down with~~
We have entered a new normal.
a pandemic h
as editorials go this is probably my most difficult
~~I had no~~.
Let me fill you in on a little secret
Sometimes the editorials [illegible]
Sometimes the idea and words just flow
from my head to my hands onto the page

Figure 1: Making a few preliminary notes

[illegible]

Rod Paddock
CODE

NEVRON OPEN VISION (NOV)

The future of .NET UI development is here

GPU Accelerated
OpenGL
Rendering

NEVRON

NOV lets you build applications for Windows, Mac and soon Blazor using a single codebase.

The suite features industry-leading UI components including Chart, Diagram, Gauge, Grid, Scheduler, Rich Text Editor, Ribbon, and others.

Nevron provides advanced solutions for:

Microsoft ASP.net | WPF | WinForms | SharePoint | Microsoft SQL Server | Xamarin

Learn more at: **www.nevron.com** today

A Simple ExpressJS and TypeScript Project

There are days I that consider myself very lucky to be a computer programmer. In the current tumult the world is going through, I'm lucky to be able to work from home and take my mind off things, things that are often very controversial and frequently depressing. So let me start this article by saying something completely non-controversial. JavaScript is awesome.

Sahil Malik

www.winsmarts.com
@sahilmalik

Sahil Malik is a Microsoft MVP, INETA speaker, a .NET author, consultant, and trainer.

Sahil loves interacting with fellow geeks in real time. His talks and trainings are full of humor and practical nuggets.

His areas of expertise are cross-platform Mobile app development, Microsoft anything, and security and identity.

I'm kidding, I'm kidding. In fact, any statement about JavaScript produces so many strong emotions on either side, it's not even funny. Since I started writing this article, there have been at least five new JavaScript packages or frameworks released.

A few days ago, a work colleague reached out to me. He happens to be a .NET programmer—a very good .NET programmer. For some crazy reason, he found himself stuck on the task of building a Web application in NodeJS.

Easier said than done, right? The problem is that in the classic Microsoft world, so many decisions are made for you. You can simply create a new .NET Core Web application project and you're ready to go. This isn't the case in the JavaScript world.

There are so many decisions to make. What version of NodeJS should you use? What node packages should you take a dependence on? JavaScript is a very flimsy language; can you take advantage of TypeScript? How do you ensure that you're following best practices? Do you want to have a productive development experience, like pressing F5 on your keyboard to run the project? And so on.

This is perhaps the biggest frustration of working with JavaScript. A simple task, such as setting up a website, has such a high initial cost and so many ways to get lost in this process that many developers never get to taste the beauty of JavaScript. But once you get over the initial gag factor, the possibilities of JavaScript are endless.

Indeed, JavaScript is awesome, or perhaps this is just my opinion. I thought I'd write an article in which I can walk you through the process of setting up a simple ExpressJS-based Web application that uses TypeScript. In writing this article, I want to ensure that I eliminate all cruft, things that might distract you, such as UI, or API, or heavy-duty frameworks. You can certainly extend what you're going to read in this article to more complicated projects. I just want to give you a simple starter project that lets you get up and running with ExpressJS and TypeScript.

If you wish to reference the code for this article, you can find it at https://github.com/maliksahil/expressjs-typescript

Create a Simple NodeJS Project

For the purpose of this article, I assume that you're somewhat familiar with NodeJS. I'm not going to spend time explaining what NodeJS is, or what yarn or npm is. Start by creating a new folder and initialize a NodeJS project using this command:

```
npm init -y
```

Running this command should give you an empty package.json file. In your package.json file, you'll have many configuration values. The three main configuration values I'm concerned about here are dependencies, devDependencies, and script.

- **dependencies** are what the project depends on. These are the node packages that the project is going to need at runtime. In other words, this is something you intend to ship.
- **devDependencies**, on the other hand, are the node packages that your project depends on, but only at coding time, not at run time. Although these node packages are very important to the project, you have no intention of shipping them in the final version.
- **scripts** are what you run and scripts can call other scripts. A typical NodeJS project is comprised of many scripts. In this case, the scripts will have the responsibility of building the project, cleaning the project, etc. Let's get to scripts in a moment, once you have the basic project structure created.

Let's start with **dependencies**. The project is going to be an ExpressJS application and a TypeScript application, and it will require, at the bare minimum, two things. It's going to need a rendering engine and it's going to need some configuration, such as what port to run on during debug time.

This is where the curse and blessing of the JavaScript world shows up. There are so many choices, and there's not one single correct answer here. For my purposes, I'm going to use "ejs" as the rendering engine, and "dotenv" as the configuration helper. I don't claim that these are the best two choices, however these are very popular and well-accepted options. I feel somewhat safe taking a dependency on these node packages. If ever in doubt, many of these node packages are open source; you should see how active their communities are, how often they merge new pull requests, how many people are involved in the project, how often they respond to the issues, how many people have starred that particular GitHub repository. These are some good baselines that you can use to ensure that you're taking dependency on a node module, that's somewhere popular, that the teams behind it are responsive enough, and that you'll have a good project lifecycle experience with that particular node package.

Given this, go ahead and modify the dependencies section of package.json as shown here:

```
"dependencies": {
  "ejs": "^3.1.3",
  "express": "^4.17.1",
  "dotenv": "^8.2.0"
},
```

Next, let's talk about **devDependencies**. I intend to write my code in TypeScript. Writing code in bare vanilla JavaScript is simply too flimsy, and experience tells me that no matter how good I get at JavaScript, I'll mess it up. TypeScript encourages me to write cleaner code, allows me to use language features that allow me to write easier-to-understand code that down-compiles to target browsers, and many more advantages.

In my devDependencies, I intend to take a dependence on TypeScript. In your devDependencies, add the following node packages:

```
"ts-node": "^8.10.2",
"tslint": "^6.1.3",
"TypeScript": "^3.9.7"
```

But wait; there's more! This alone isn't enough. When you start writing code in TypeScript, you take on two new responsibilities. TypeScript transpiles into JavaScript, and that JavaScript is what you finally run. You don't usually run TypeScript directly. I say "usually" because sometimes for some helper files, it may be okay to transpile on the fly and just run it. For the code you care about, your website, you'll always statically transpile this code. You don't want to transpile it on the fly, because it'd be just too slow. Your first responsibility is all the paraphernalia associated with transpiling.

There's one more thing you need to worry about. Now that you're using TypeScript, TypeScript will enforce some rules on you. TypeScript wishes to make sure that you're using the correct data types, for example. This is a very good thing because without the appropriate datatypes, you don't get the correct IntelliSense. Without the correct IntelliSense, you're more prone to making mistakes. Luckily, there are a number of types node packages that give you all these data types that you care about. In this project, you're going to take a dependency on those as well. Given all this, you can see the full devDependencies in **Listing 1**.

Let's take a moment to understand **Listing 1**. The various @types node packages that you see are the type definitions that the TypeScript transpiler will depend on, and the IDE, such as VSCode, will depend on to show the correct IntelliSense.

The **shelljs node package** gives you shell-like commands that you can use in TypeScript. You'll use this package to copy transpiled files into a folder from where the Web application can be served.

The **nodemon** node package allows you to restart your application if the underlying files change. I'll use this for debug purposes, the idea being, that as I edit a file, the build will automatically run and allow me to see changes in the browser. The **npm**-run-all allows you to run multiple npm scripts in parallel, something you'll have to do when you build your site. The **rimraf** allows you to delete a bunch of files easily and you'll use this to clean your project. The final three, **ts-node**, **tslint**, and **TypeScript** are for TypeScript support, as I mentioned above.

The final section to talk about is the **scripts** section. I'm not ready to talk about that yet. Let's finish more of the project first so you have a basic structure in place, and then I'll talk about how scripts allows you to work with the project structure.

Configuration Settings

Before you can start working on your ExpressJS application, let's define the environment and the various configuration details that ExpressJS application depends on.

The first thing you need is an environment file, which includes the basic configuration settings, such as what port the application will run on. Because I've have taken a dependency on dotnev for this, I simply create a .env file in the root of the project with the following contents:

```
SERVER_PORT=8080
HOST_URL=http://localhost:8080
```

It's customary to add this .env to your .gitignore and check in a .env.sample file instead. While I'm at it, here's my full .gitignore:

```
.cache/
.vscode/
dist/
node_modules/
.env
.DS_Store
*.log
```

Listing 1: The devDependencies section of the package.json

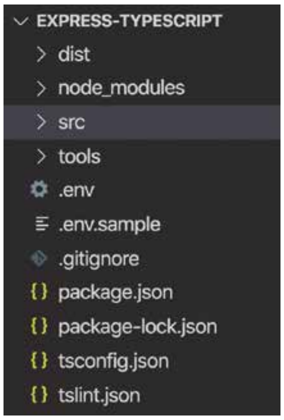
```
"devDependencies": {
  "@types/dotenv": "^8.2.0",
  "@types/express": "^4.17.7",
  "@types/node": "^14.0.27",
  "@types/shelljs": "0.7.9",
  "shelljs": "0.8.4",
  "nodemon": "^2.0.4",
  "npm-run-all": "^4.1.5",
  "rimraf": "^3.0.2",
  "ts-node": "^8.10.2",
  "tslint": "^6.1.3",
  "TypeScript": "^3.9.7"
},
```

Listing 2: The tsconfig.json file

```
{
    "compilerOptions": {
        "module": "commonjs",
        "esModuleInterop": true,
        "target": "es6",
        "noImplicitAny": true,
        "moduleResolution": "node",
        "sourceMap": true,
        "outDir": "dist",
        "baseUrl": ".",
        "paths": {
            "*": [
                "node_modules/*",
                "src/types/*"
            ]
        }
    },
    "include": [
        "src/**/*"
    ],
    "exclude": [
        "src/public"
    ]
}
```

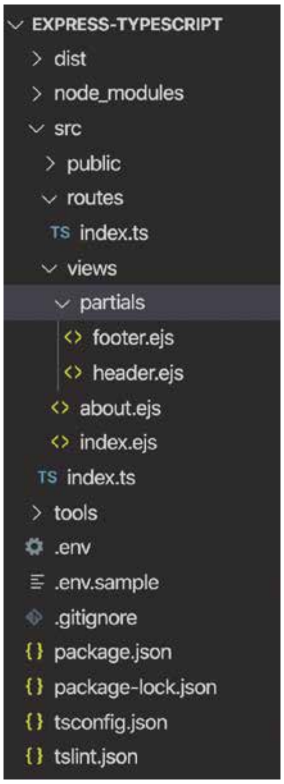
Listing 3: My tslint.json

```
{
    "defaultSeverity": "error",
    "extends": [
        "tslint:recommended"
    ],
    "jsRules": {},
    "rules": {
        "indent": [ true, "spaces" ],
        "trailing-comma": [ false ]
    },
    "rulesDirectory": []
}
```



```
EXPRESS-TYPESCRIPT
  > dist
  > node_modules
  > src
  > tools
  .env
  .env.sample
  .gitignore
  package.json
  package-lock.json
  tsconfig.json
  tslint.json
```

Figure 1: Our basic project structure



```
EXPRESS-TYPESCRIPT
  > dist
  > node_modules
  v src
    > public
    v routes
      index.ts
    v views
      v partials
        footer.ejs
        header.ejs
      about.ejs
      index.ejs
    index.ts
  > tools
  .env
  .env.sample
  .gitignore
  package.json
  package-lock.json
  tsconfig.json
  tslint.json
```

Figure 2: Full project structure

As you can see, this is a very barebones project. I intend to show you every bit of code I'll write here.

Next, let's author the tsconfig.json file. The tsconfig.json file informs the TypesScript transpiler of the various settings you want it to use. You can see my full tsconfig.json in **Listing 2**. As you can see in **Listing 2**, this is a fairly simple tsconfig.json. I'm targeting commonjs and excluding paths I don't want compiled. Note that I've also given a hint to include node_modules and src/types, without which the TypeScript transpilation would fail.

Finally, let's author a tslint.json file. The purpose of the tslint.json file is to author all of the linting rules. Even though TypeScript encourages you to write good code, good is only as good as the team or the developer. Additionally, a lot of things are opinions or arbitrary decisions. For example, what character should you use for a quote? Should it be ' or "? Both are correct, but it's important to pick one and stick with it. You can specify a lot of rules in tslint.json, but for my purposes, I'll keep it simple. You can see my tslint.json in **Listing 3**.

While we're at it, add a linting script to your package.json as well, as follows:

```
"lint":
 "tslint -c tslint.json -p tsconfig.json"
```

This script runs the linter and throws an error if your written code breaks one of the linting rules. Imagine if I were debugging and writing code in parallel, if any time I wrote poor code, as long as the linter runs inline of my build process, I'll be alerted immediately. I'll show later in this article how you can integrate this REPL pipeline. You can also integrate this as a part of your check-in/branch merge rules, but I won't be covering that in this article.

Write Your Application

So far, you have your configuration settings and your package dependencies done. You also added one linting script. Now, let's focus on the main project structure. Again, NodeJS is not very opinionated; you can make this as clean or as messy as you'd like. As I think about creating this project structure, I have three main things I'd like to achieve.

- I'd like a **src** folder where my source code lives. This is where I write my ExpressJS application in TypeScript.
- I'd like a **dist** folder, where the built version of this application will live. This is what will eventually run.
- I'd like a **tools** folder, where I will have helper scripts to facilitate the build process or anything else I might need.

At this point, go ahead and create these three folders in your project structure. Your project should look like **Figure 1** at this point.

Let's tackle the easier part first. The dist folder is going to be empty. It's just a place holder where my "built" artifacts will go. Note that I have also specified in my .gitignore file that the contents of this folder are not to be checked in.

Next, let's talk about the src folder. This is where my ExpressJS application will go. There are many ways to structure an ExpressJS app, but I've written my application so that it has a couple of routes, and those couple of routes are backed by a couple of views. My views are called index and about, served at "/" and "/about" routes. Both of these views rely on partial views for the header and footers.

Additionally, I've also created a "public" folder, where artifacts that don't need transpilation can sit. This could be a third-party JavaScript library or framework, such as Bootstrap.

At this point, my project structure looks like **Figure 2**.

This isn't a tutorial on ExpressJS and you could replace ExpressJS with really any JavaScript framework intended to serve Web applications here. But, let's understand at a high level how the application is built.

You can see my index.ts in **Listing 4**. In **Listing 4**, I'm using ejs as my view engine and I'm rendering views out of the "views" folder. Additionally, I'm serving static files from the public folder, and registering routes from the routes folder. The routes folder serves two routes: index, and about.

The routes.ts file can be seen in **Listing 5**. I'm using a concept called barreling, where from index.ts, I can simply import the routes folder. TypeScript looks for a file called index.ts in a folder called routes and loads everything accordingly. This makes it really convenient to separate concerns into folders as my application grows.

Finally, my views are rather simple. As can be seen from **Figure 2**, I have two views, index and about. These match with what you see in **Listing 5**, where I've defined my routes. This could be made dynamic, if you wish. My index.ejs view looks like **Listing 6**. I'll omit the about.ejs view for brevity, although it's very similar to index.ejs. Hey, it's just a simple Web page that says "About". You're welcome to check out the code in the GitHub repo mentioned above.

The partial views are a reusable snippet of ejs views that I can embed in any other such view. As an example, the header.ejs is shown here:

```
<h1>Header</h1>
<a href="/">Home</a> |
<a href="/about">About</a>
```

Again, you can make this application a lot more complex if you wish. This is a great start and you can build from here using this as a template.

Now, let's focus your attention on the tools folder.

You're going to need some mechanism to build the application during debug time. One part of the build is the Type-

Script transpilation, but the second part is copying over the static assets, such as CSS, JavaScript, images, etc., that don't need transpilation. You're going to put all of those in the public folder. These files need to be copied from the src folder and be copied into the dist folder at runtime. To facilitate that, go ahead and create a file called copyAssets.ts in the tools folder. You can find the code for copyAssets.ts in **Listing 7**. As you can see from **Listing 7**, I'm simply cleaning up and copying files as the application runs.

Run and Build Your Application

Now that the application is done, it's time to stitch all of this together and make it work during debug time. Deploying into production is a whole other topic, and there are many ways to do that as well. For example, you could easily convert the dist folder into a Docker image. I hope to talk more about deployment and other concerns in future articles. For now, let's get the application working in debug mode.

To begin, add a "main" entry in the package.json, as shown:

```
"main": "dist/index.js",
```

This informs the host that when this project is asked to run, you should run the dist/index.js file. For instance, if you press F5 in VSCode, this file will run. Of course, this file doesn't yet exist; it has to be built. I'll get there in a moment. For now, in your scripts tag, add another script to run this main entry point:

```
"dev": "node .",
```

Now, let's add a few helper scripts in the scripts section of the package.json. You'll use these helper scripts as building blocks to facilitate the debugging experience. These can be seen here:

```
"copy-assets": "ts-node tools/copyAssets",
"tsc": "tsc",
"clean": "rimraf dist/*",
"lint":
 "tslint -c tslint.json -p tsconfig.json"
```

The copy-assets script runs the copyAssets.ts file using the ts-node package. If you remember, this file copies the static content that the site depends on and puts it in the dist folder. The tsc script represents the TypeScript transpiler.

The clean script is used to delete older files. The idea is that as I change code, I want my application to be automatically rebuilt and recopied. Before it's recopied, I want older applications to be deleted. That's what I'm using rimraf and clean for.

And the lint script, as I have already described in this article, is linting the code to ensure that I'm writing clean code.

So, what would the build look like? Simple: clean, lint, run the TypeScript transpiler to generate files and then run copy-assets. Go ahead and add a build script as below.

```
"build":
 "npm-run-all clean lint tsc copy-assets",
```

Listing 4: My index.ts

```
import dotenv from "dotenv";
import express from "express";
import path from "path";
import * as routes from "./routes";

dotenv.config();

const port = process.env.SERVER_PORT;
const app = express();

app.use(express.json());

app.set("views", path.join(__dirname, "views"));
app.set("view engine", "ejs");

app.use(express.static(path.join(__dirname, "public")));

// Configure routes
routes.register(app);

// start the express server
app.listen(port, () => {
    // tslint:disable-next-line:no-console
    console.log(
 `server started at http://localhost:${port}`);
});
```

Listing 5: The routes.ts file

```
import * as express from "express";

export const register = (app: express.Application)
   => {
    // home page
    app.get("/", (req: any, res) => {
        res.render("index");
    });

    // about page
    app.get("/about", (req: any, res) => {
        res.render("about");
    });
};
```

Listing 6: Index.ejs

```
<!doctype html>

<html lang="en">

<head>
    <meta charset="utf-8">
    <title>ExpressJS TypeScript starter template</title>
</head>

<body>
    <%- include('partials/header') %>
    <h1>Home Page</h1>
    <%- include('partials/footer') %>
</body>

</html>
```

Listing 7: copyAssets.ts

```
import * as shell from "shelljs";

// Copy all the view templates and assets in the public folder
shell.cp("-R", ["src/views", "src/public"], "dist/");

// Remove unnecessary files
shell.rm(["dist/public/js/*.ts", "dist/public/js/*.json"]);
```

Listing 8: The script section

```
"scripts": {
  "start":
  "nodemon --watch src -e
    ts,ejs --exec npm run dev:start",
  "dev": "node .",
  "dev:start": "npm-run-all build dev",
  "build": "npm-run-all clean lint tsc copy-assets",
  "copy-assets": "ts-node tools/copyAssets",
  "tsc": "tsc",
  "clean": "rimraf dist/*",
  "lint": "tslint -c tslint.json -p tsconfig.json"
},
```

SPONSORED SIDEBAR:

Interested in Moving to Azure? CODE Can Help!

Take advantage of a FREE hour-long CODE Consulting session (yes, FREE!) to jumpstart your organization's plans to develop solutions on the Microsoft Azure platform. For more information, visit www.codemag.com/consulting or email us at info@codemag.com.

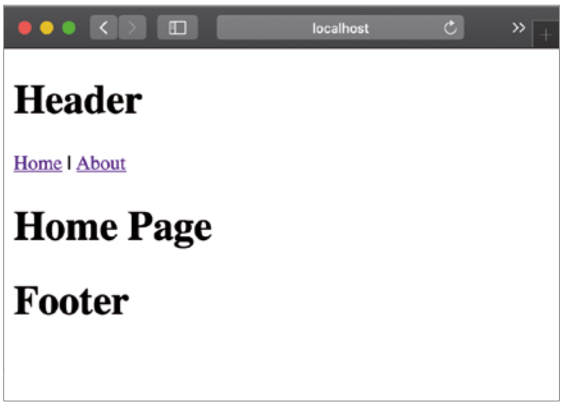


Figure 3: Your application is running.

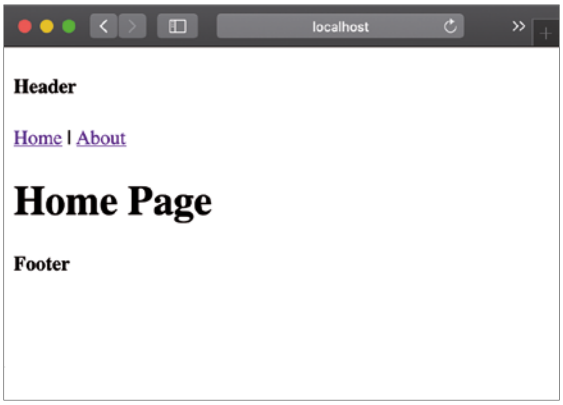


Figure 4: Your application reflects changes immediately.

Now you can build. What's next? Let's build and run! You already have a script entry called "dev" that runs the project. Now you need to run build and dev together. Here's how you do it:

```
"dev:start": "npm-run-all build dev",
```

At this point, if I were to write npm run dev:start, my application will build and run. But that's not enough! What I really want is for my application to automatically build, rebuild, deploy, and run as I edit code. To facilitate that, add the "start" command as below.

```
"start":
"nodemon --watch src -e
  ts,ejs --exec npm run dev:start",
```

This finishes the "script" section for the project. You can find the full script section in **Listing 8**.

Run Your Application

Now for the most fun part. To run the application, type the following command at terminal:

```
npm start
```

Sure, it's possible to wire up VSCode, so pressing F5 will do this for you. But let's, for now, run it from commandline and get started. Once your application is built and deployed, you should see the following in your console:

```
server started at http://localhost:8080
```

Now open your favorite browser and visit http://localhost:8080. Your application can be seen running in **Figure 3**.

This looks great, but the header and footer are HUGE. Let's make them smaller. As the application is running, edit the src\public\views\partials\header.ejs and footer.ejs files, and change the h1 tag to h4. You'll notice that soon as you save the file, the application rebuilds automatically. Now visit your browser and press F5 to see the changes.

You have a nice REPL (read-eval-print-loop)-based development environment set up.

Summary

In this article, I showed you every bit, corner, and nuance of writing an ExpressJS and TypeScript-based project. Let me just say this: As technology is moving forward very fast, the one thing that suffers is documentation. All of these technologies are amazing, and once you master them all, you are incredibly productive. The issue, of course, is mastering them, because no single article walks you through the very basics, no single article shows you every single step you need to take to get a basic application running.

This can be incredibly frustrating for beginners. And let's be honest. In our industry, all of us are always beginners.

There's plenty more that can be added into this project, of course. For example, can I build a compelling user interface using Bootstrap? Can I add authentication? Can I add something like a browser link, so when my application rebuilds, I don't have to manually press F5 in development mode? How do I deploy this application? Can I easily convert this into a Docker image? And when I convert this into a Docker image, what if the node process crashes in the Docker image? How do I gracefully recover from that? Can I use something like Kubernetes to automatically check the health of my code, and recover gracefully? How do I add APIs into my project? And so much more. All of these concerns are what you will deal with in a real-life project.

I hope to talk about these and many more things while enhancing this project template in my future articles. Until then, happy coding.

Sahil Malik

CODE
LEGACY

OLD
TECH HOLDING
YOU BACK?

Are you being held back by a legacy application that needs to be modernized? We can help.
We specialize in converting legacy applications to modern technologies. Whether your application is currently written in Visual Basic, FoxPro, Access, ASP Classic, .NET 1.0, PHP, Delphi…
or something else, we can help.

codemag.com/legacy
832-717-4445 ext. 9 • info@codemag.com

Using Geolocation and Google Maps

As many users browse websites on their mobile phones, you might need the ability to guide the user from their current location to your location. This is easily accomplished using the browser's built-in navigator.geolocation object and Google Maps. The geolocation object provides the latitude and longitude of the user's phone or desktop. You can embed a Google map on your Web page and show the user their location based on that latitude and longitude. Additional API calls to Google's mapping API can give the user step-by-step directions to your location. This article shows you how to get started using these two powerful APIs.

Paul D. Sheriff
psheriff@pdsa.com
http://www.pdsa.com

Paul has been in the IT industry over 33 years. In that time, he has successfully assisted hundreds of companies architect software applications to solve their toughest business problems. Paul has been a teacher and mentor through various mediums such as video courses, blogs, articles, and speaking engagements at user groups and conferences around the world. Paul has 26 courses in the www.pluralsight.com library (http://www.pluralsight.com/author/paul-sheriff) on topics ranging from JavaScript, Angular, MVC, WPF, XML, jQuery, and Bootstrap.

Create HTML to Display Location Information

To begin, build a simple HTML page to display the various properties of the geolocation object. Feel free to use your tool(s) of choice to build an HTML page named **index.html** and a Web project. You can use Node.js/Express, MVC, Web Forms, etc. All you need is a Web server to run the HTML page and a CDN reference to jQuery. After creating your Web project and the index.html page, open the index.html file and add the code shown in **Listing 1** into this file.

Styles

The index.html file references a stylesheet named site.css. In this file is where you place some CSS rules to set some margins, padding, and some color. Add a file named **site.css** in a **styles** folder (add this folder if needed) and place the code shown below into this file.

```
header, main { padding-left: 2em; }

.alert {
  margin-top: 1em;
  margin-bottom: 1em;
  padding: 1em;
  background-color: red;
  color: white;
}

.d-none { display: none; }
```

Get Latitude and Longitude

When using the geolocation object, the first thing you should do is check to ensure that the user's browser supports the object. After your page loads, check to see if something other than a null or undefined is returned when you check the navigator.geolocation property. If a geolocation object is returned, you can call the getCurrentPosition() method on that object and pass in the name of a function to call once the current location has been determined. If a geolocation object isn't returned, call a function to display an error message. Add the code shown below within the $(document).ready() function.

```
$(document).ready(function () {
  if (navigator.geolocation) {
    navigator.geolocation.
      getCurrentPosition(displayPosition);
  }
  else {
    displayError("Please update your browser
                  to use Geolocation.");
  }
});
```

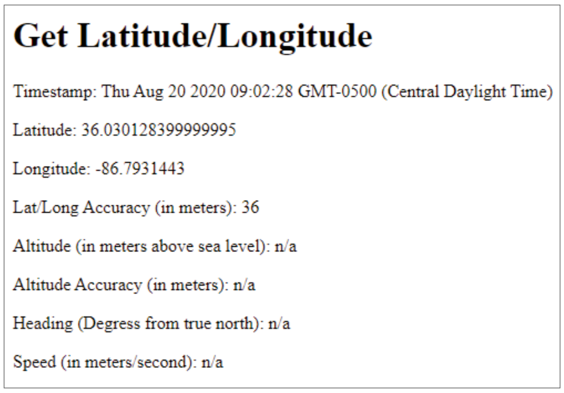
Display Latitude and Longitude

The getCurrentPosition() method asks the user's permission to retrieve their location. The browser prompts the user to answer with an "OK or "Yes" if they will allow the getCurrentPosition() method to retrieve their current location. If the user answers in the affirmative, the user's location is retrieved, and the data is placed into a **Coordinates** object. This Coordinates object is passed to the callback function you passed into the getCurrentPosition() method. Within the displayPosition() function is where you extract the latitude, longitude, and other properties. Add the following code within the displayPosition() function.

```
function displayPosition(pos) {
  let coords = pos.coords;

  $("#timestamp").text(new Date(pos.timestamp));
  $("#lat").text(coords.latitude);
  $("#long").text(coords.longitude);
  $("#accuracy").text(coords.accuracy);
  $("#altitude").text(coords.altitude ?? "n/a");
  $("#altitudeaccuracy").
    text(coords.altitudeAccuracy ?? "n/a");
  $("#heading").text(coords.heading ?? "n/a");
  $("#speed").text(coords.speed ?? "n/a");
}
```

Try It Out

Save the changes to all your files and run your Web project. Your browser prompts you that this Web page is asking to know your location. There will be two options: to allow this page to access the geolocation coordinates or decline access. For now, go ahead and allow access to see your latitude, longitude, and other properties. You should see data appear like the data shown in **Figure 1**.



Get Latitude/Longitude

Timestamp: Thu Aug 20 2020 09:02:28 GMT-0500 (Central Daylight Time)

Latitude: 36.030128399999995

Longitude: -86.7931443

Lat/Long Accuracy (in meters): 36

Altitude (in meters above sea level): n/a

Altitude Accuracy (in meters): n/a

Heading (Degress from true north): n/a

Speed (in meters/second): n/a

Figure 1: Display Latitude, Longitude, and Accuracy

Listing 1: Create an HTML page to display the geolocation coordinate information.

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8" />

  <title>Get Latitude/Longitude</title>

  <link href="styles/site.css" rel="stylesheet" />
</head>

<body>
  <header>
    <h1>Get Latitude/Longitude</h1>
  </header>

  <main>
    <p>Timestamp:
      <span id="timestamp"></span>
    </p>
    <p>Latitude:
      <span id="lat"></span>
    </p>
    <p>Longitude:
      <span id="long"></span>
    </p>
    <p>Lat/Long Accuracy (meters):
      <span id="accuracy"></span>
    </p>
    <p>Altitude (meters above sea level):
      <span id="altitude"></span>
    </p>
    <p>Altitude Accuracy (meters):
      <span id="altitudeaccuracy"></span>
    </p>
    <p>Heading (Degress from true north):
      <span id="heading"></span>
    </p>
    <p>Speed (meters/second):
      <span id="speed"></span>
    </p>

    <div id="errorArea"
         class="alert d-none">
    </div>
  </main>

  <script src="https://code.jquery.com/
               jquery-3.5.1.min.js"
    integrity="REMOVED FOR BREVITY"
    crossorigin="anonymous">
  </script>
  <script>
    'use strict';

  $(document).ready(function () {
  }

  function displayPosition(pos) {
  }

  function handleError(error) {
  }

  function displayError(msg) {
  }
  </script>
</body>

</html>
```

Add Error Handling

You should not assume that the geolocation object will always work. There are a few things that can go wrong: the user may not allow access to the geolocation object, the call to getCurrentPosition() may timeout, the current user's location may not be able to be determined, and other errors may occur. In case an error does occur, make sure to set up an error message to display on the page.

In the index.html page you created, there is a <div id="errorArea"> where you place any error messages. This <div> has a style of "alert," which displays error messages with a red background and white letters. The style of "d-none" is also in the class attribute. This hides the <div> until you're ready to display an error message. Add the following code in the displayError() function.

```
function displayError(msg) {
  $("#errorArea").removeClass("d-none");
  $("#errorArea").html(msg);
}
```

Handle Location Errors

The second parameter you may pass to the getCurrentPosition() method is a callback function to handle any errors that may occur in that method. Add the code shown in **Listing 2** to the handleError() function stub you created earlier.

Pass in the name of the handleError() function as the second parameter in the call to the getCurrentPosition() method.

```
if (navigator.geolocation) {
  navigator.geolocation.getCurrentPosition(
    displayPosition, handleError);
}
```

Listing 2: Handle location errors by displaying a specific error message.

```
function handleError(error) {
  let msg = "";

  console.log("error.message = " + error.message);
  switch (error.code) {
    case error.PERMISSION_DENIED:
      msg = "User does not want to display location.";
      break;
    case error.POSITION_UNAVAILABLE:
      msg = "Can't determine user's location.";
      break;
    case error.TIMEOUT:
      msg = "The request for geolocation info timed out.";
      break;
    case error.UNKNOWN_ERROR:
      msg = "An unknown error occurred.";
      break;
  }

  displayError(msg);
}
```

Try It Out

Run the page again, but this time when the browser prompts you to use geolocation, answer in the negative. You should now be presented with a page that looks like **Figure 2**.

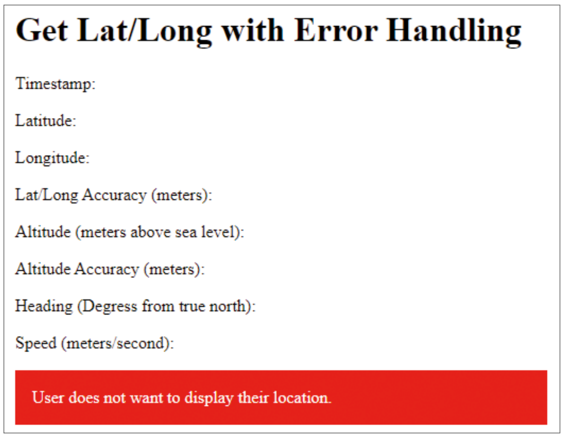
Geolocation Options

There's a third parameter you may pass to the getCurrentPosition() method. This third parameter is an optional

PositionOptions literal object. This object contains three properties and you may set one, two, or all three of them. One thing to note: these options are only applied if you're running in HTTPS mode in your browser. The properties and their meanings are listed in **Table 1**.

Property	Description
enableHighAccuracy	If set to true, informs the device to attempt to obtain a more accurate position. This can slow down the time it takes to retrieve the coordinates.
timeout	The maximum length of time to wait before throwing an exception. This value is expressed in milliseconds.
maximumAge	The maximum age of the last cached position that's acceptable to return. This value is expressed in milliseconds. Set to 0 to not allow the device to cache positions.

Table 1: You may set any or none of the PositionOptions.



Get Lat/Long with Error Handling

Timestamp:

Latitude:

Longitude:

Lat/Long Accuracy (meters):

Altitude (meters above sea level):

Altitude Accuracy (meters):

Heading (Degress from true north):

Speed (meters/second):

User does not want to display their location.

Figure 2: Create an error area to display error messages

Listing 3: Handle location errors by displaying a specific error message.

```
'use strict';

let geoController = (function () {
  // ****************************
  // Private Variables
  // ****************************
  let position = null;
  let lastMessage = null;
  let lastError = null;
  let successCallback = null;
  let errorCallback = null;

  // Enable high accuracy, if available
  // Timeout after 10 seconds
  // Only cached positions where age < 5 minutes
  let options = {
    enableHighAccuracy: true,
    timeout: 10000,
    maximumAge: 300000
  };

  // ****************************
  // Private Functions
  // ****************************
  function getCurrentPosition(success, error, posOptions) {
    // Set callbacks
    successCallback = success;
    errorCallback = error;
    if (posOptions) {
      options = posOptions;
    }

    // Reset private variables
    position = null;
    lastError = null;
    lastMessage = null;

    if (navigator.geolocation) {
      navigator.geolocation.getCurrentPosition(
        setPosition, handleError, options);
    }
    else {
      callError("Update your browser to use Geolocation.");
    }
  }

  function setPosition(pos) {
    position = pos;

    if (successCallback) {
      successCallback(position);
    }
  }

  function handleError(error) {
    lastError = error;

    switch (error.code) {
      case error.PERMISSION_DENIED:
        lastMessage = "User does not want
          to display their location."
        break;
      case error.POSITION_UNAVAILABLE:
        lastMessage = "Can't determine
          user's location."
        break;
      case error.TIMEOUT:
        lastMessage = "The request for
          geolocation information timed out."
        break;
      case error.UNKNOWN_ERROR:
        lastMessage = "An unknown error occurred."
        break;
    }

    callError(lastMessage);
  }

  function callError(msg) {
    lastMessage = msg;

    console.log(msg);

    if (errorCallback) {
      errorCallback(lastMessage);
    }
  }

  // ****************************
  // Public Functions
  // ****************************
  return {
    "getCurrentPosition": function (success,
                           error, posOptions) {
      getCurrentPosition(success, error,
        posOptions);
    },
    "getPosition": function () {
      return position;
    },
    "getLastError": function () {
      return lastError;
    },
    "getLastMessage": function () {
      return lastMessage;
    },
    "getOptions": function () {
      return options;
    }
  }
})();
```

To use these options, add a new variable named **options**, and set it equal to a literal object you see in the code in bold below. Pass this object as the third parameter to the getCurrentPosition() method.

```
if (navigator.geolocation) {
  let options = {
    enableHighAccuracy: true,
    timeout: 10000,
    maximumAge: 300000
  };

  navigator.geolocation.getCurrentPosition(
    displayPosition, handleError, options);
}
```

Create a Geolocation Closure

The code shown so far is fine for this page but requires you to copy and paste the code and make significant changes if you want to use this code on multiple Web pages. To make the geolocation code more reusable, wrap it into a closure and place that closure in a JavaScript file. Create a file named **geoController.js** in a **scripts** folder (create this folder if needed) and add the code shown in **Listing 3**.

The basics of this closure are created using an Immediately Invoked Function Expression, more commonly called an IIFE. This IIFE starts with the second line in the file and ends with the last line of the file. In between are some private variables, private functions, and a literal object returned from this IIFE to identify the public functions to be called from the **geoController** variable, as shown in **Listing 4**.

The only private function that's exposed as a public function is getCurrentPosition(). The rest of the functions allow you to retrieve values set in the private variables such as the position object, the last error object, the last error message, and the position option object. The getCurrentPosition() function has the same signature as the getCurrentPosition() method on the geolocation object to make it easy to understand. The two callbacks are setPosition() and handleError() within this closure. This allows you to set the private variables from these two functions before calling the success or error callbacks you pass in.

Notice that in the closure, there's no code that affects the UI. All this function does is set data. You can then call any of the public functions to retrieve the data and use that to set the UI on your specific page. Open the **index.html** page and change the code at the bottom of the file to look like **Listing 5**.

In the code in **Listing 5**, you see the call to the geoController.getCurrentPosition() function. Pass in two function names on this page as the callbacks to be used to set the position data into the page, and to display an error message. You now have a nice separation of concerns as far as code that gathers data and code that displays data.

Pass Coordinates to Google Maps

Once you have the latitude and longitude from the geolocation object, you have a variety of options on how to display those coordinates. There are free and paid services you can use. For example, with Google Maps, Bing Maps, or Open Street Maps, you can redirect to their Web pages and pass in the coordinates. You can do this for free. Using Google Maps or Bing Maps, you also have the choice of paying for their service and having a map embedded right into your own Web page. Let's take a look at the free option first. Open the **index.html** page and add a button within the <main> element.

```
<button onclick="displayOnMap();">
  Display on Map
</button>
```

Add a function within the <script> tags named displayOnMap() as shown in the code below.

```
function displayOnMap() {
  let coords = geoController
```

Listing 4: The structure of your closures should follow this format.

```
let geoController = (function () {
  // Private variables here

  // The following are private functions
  function getCurrentPosition(success,
           error, posOptions) {
  }

  function setPosition(pos) {
  }

  function handleError(error) {
  }

  function callError(msg) {
  }

  return {
    // Public functions here
  }
})();
```

Listing 5: Call the closure to set data, then make a call back to your page to affect the UI.

```
<script src="scripts/geoController.js"
        type="text/javascript">
</script>
<script>
  'use strict';

  $(document).ready(function () {
    geoController.getCurrentPosition(
      displayPosition, displayError);
  });

  function displayPosition(pos) {
    let coords = pos.coords;
    $("#timestamp").text(new Date(pos.timestamp));
    $("#lat").text(coords.latitude);
    $("#long").text(coords.longitude);
    $("#accuracy").text(coords.accuracy);
    $("#altitude").text(coords.altitude ?? "n/a");
    $("#altitudeaccuracy")
      .text(coords.altitudeAccuracy ?? "n/a");
    $("#heading").text(coords.heading ?? "n/a");
    $("#speed").text(coords.speed ?? "n/a");
  }

  function displayError(msg) {
    $("#errorArea").removeClass("d-none");
    $("#errorArea").html(msg);
  }
</script>
```

Listing 6: Use the DirectionsService and DirectionsRenderer classes to get directions between two points.

```
function addDirections(from, to) {
  let service =
    new google.maps.DirectionsService();
  let renderer =
    new google.maps.DirectionsRenderer();

  // Set route of how to travel
  // from point A to B
  service.route(
    {
      origin: from,
      destination: to,
      travelMode: 'DRIVING'
    },
    function (response, status) {
      if (status === 'OK') {
        renderer.setDirections(response);
        // Render directions on the map
        renderer.setMap(map);
      } else {
        console.log(
          'Directions request failed due to '
            + status);
      }
    });
}
```

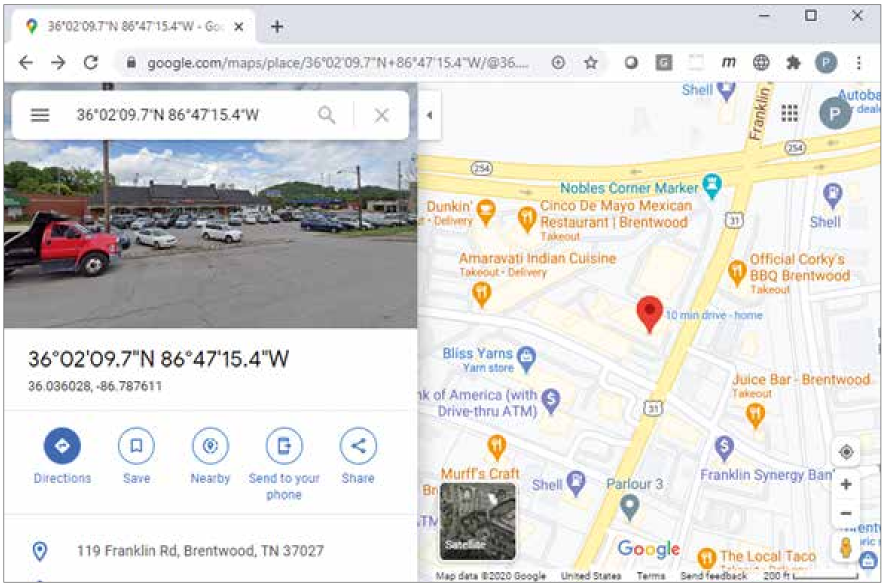

36°02'09.7"N 86°47'15.4"W - Go
google.com/maps/place/36°02'09.7"N+86°47'15.4"W/@36....
36°02'09.7"N 86°47'15.4"W
36.036028, -86.787611
Directions
Save
Nearby
Send to your phone
Share
119 Franklin Rd, Brentwood, TN 37027
Nobles Corner Market
Cinco De Mayo Mexican Restaurant | Brentwood
Amaravati Indian Cuisine
Official Corky's BBQ Brentwood
Bliss Yarns
Juice Bar - Brentwood
Murff's Craft
Parlour 3
Franklin Synergy Bank
The Local Taco
Shell
Satellite
Google

Figure 3: Google Maps can display any latitude and longitude through the URL.

```
    .getPosition().coords;

  window.location.href =
    `https://google.com/maps?
    q=${coords.latitude},
         ${coords.longitude}`;
}
```

This code retrieves the **GeoLocationPosition** object by calling the getPosition() method in your closure. It then extracts the *coords* property and puts it into a local variable named **coords**. Pass in the latitude and longitude on the URL line in a format that looks like the following:

```
https://google.com/maps?q=36.084391,-86.773550
```

The code in displayOnMap() sets the window.location.href property using a template string to embed the latitude and longitude retrieved from the **coords** variable. Once the **href** property is set, your browser redirects to Google Maps and displays a marker at those coordinates, as shown in **Figure 3**.

Of course, the problem with this approach is that you must leave your site and go to the Google Maps site. A better approach is to embed a Google Map within your Web page. The next part of this article covers how to accomplish this.

Embed a Google Map

Simplify the <main> element by removing most of the <p> and <span> elements except the ones used to display latitude and longitude. Add a new <div> element just above the <div id="errorArea"> and add *id* and *class* attributes to the new <div> element, as shown in the code snippet below.

```
<main>
  <p>Latitude: <span id="lat"></span></p>
  <p>Longitude: <span id="long"></span></p>

  <div id="map" class="mapArea"></div>

  <div id="errorArea" class="alert d-none">
  </div>
</main>
```

Open the **styles\site.css** file and add a new CSS rule for the *mapArea* class. Add the CSS rule shown below to set the width and height of the <div> where the map is to be embedded.

```
.mapArea {
  width: 90%;
  height: 500px;
}
```

Fix the Script Code

Remove the $(document).ready() function and replace it with a function named initialize().

```
function initialize() {
  geoController.getCurrentPosition(
    displayPosition, displayError);
};
```

Locate the displayPosition() function and remove the code that set the elements you removed from the HTML. Also, make a call to a new function named addMap().

```
function displayPosition(pos) {
  let coords = pos.coords;
  $("#lat").text(coords.latitude);
  $("#long").text(coords.longitude);

  addMap(coords);
}
```

Add a new function named addMap() just below the displayPosition() function. This method looks like the following:

```
function addMap(location) {
  // Create a lat/lng object
  let pos = new google.maps.LatLng(
     location.latitude, location.longitude);
  // Create map options
  let mapOptions = {
    center: pos,
    zoom: 16,
    mapTypeId: google.maps.MapTypeId.ROADMAP
  };
  // Create new google map
  let map = new google.maps.Map(
    document.getElementById("map"),
    mapOptions);
}
```

The addMap() function builds a new **LatLng** object by passing in the latitude and longitude properties from the geolocation coordinates retrieved from the call to getCurrentPosition(). Next, create a map options object with the properties **center**, **zoom** and **mapTypeId**. Set the **center** property to the **LatLng** object so the map knows where to put the coordinates on the map. Set the **zoom** property to 16 so the API can show just a certain portion of the map. You might want to play with this property to get make it look the way you want. The **mapTypeId** property is a constant that tells the type of map to display. Valid values are roadmap, satellite, hybrid, or terrain. The last piece of code is to create a **Map** object by passing in the HTML object where you wish to embed the map and the map options object.

Create a Closure for Mapping

Just like you created a closure for working with geolocation, I recommend that you use a closure on each HTML page as well. Immediately after the **'use strict'**, add the code shown below.

```
<script>
  'use strict';

  var gmapController = (function () {

    return {
      "initialize": initialize
    }
  })();

  // Rest of the code here
</script>
```

Take all of the functions that were there before and move them within the closure you just created. By moving all of the code within the function, each of those functions becomes private within the **gmapController** function. The only function that needs to be made public from the closure is the initialize() function, so that's the one you add to the literal object returned from the closure.

Call the Initialize Method

Because you removed the $(document).ready() function and moved all of the code within a closure, how does the map get displayed? There's one more piece to this puzzle that you add to your page. Put the following <script> tag immediately after the ending </script> tag where your code is located.

```
<script
  src="https://maps.googleapis.com/
       maps/api/js?key=YOUR_KEY_HERE
       &callback=gmapController.initialize"
  type="text/javascript">
</script>
```

In the above code, you need to add a developer API key, which is covered in the next section of this article. The "callback=gmapController.initialize" is passed into the Google JavaScript and tells the API that after it has loaded all its code, it should call the initialize() function within the closure on this page. This's how you get your map embedded on your page.

Get a Google API Key

Before you can embed a Google map, you must sign up with Google to get an API key at https://tinyurl.com/y5reu2l2. Be sure to read the instructions on this page carefully, as you should turn on various API services based on what you wish to use. In addition, you should also restrict the usage of your API key to only a few applications. You might also want to set daily or monthly limits on the amount that can be spent to ensure that you don't blow out your budget.

Although it doesn't cost anything to obtain a Google API key, you must enter a credit card. At the time of this writing, you get $200 per month of their various mapping APIs free. This amount should be more than enough to get your maps working within your website.

ADVERTISERS INDEX

Advertisers Index

CODE Consulting	www.codemag.com/consulting	41
CODE Legacy	www.codemag.com/legacy	13
CODE Magazine	www.codemag.com/subscribe	75
DevIntersection	www.devintersection.com	2
dtSearch	www.dtSearch.com	29
Fortellis	www.fortellis.io	76
LEAD Technologies	www.leadtools.com	5
Nevron Software, LLC	www.nevron.com	7

Geolocation, Vue, .NET Tooling, ExpressJS
CODE
Ready, Set, Building Applications with GO!
Using the .NET Core Tools Framework
Building Geolocation-Enabled Applications
Getting Started with ExpressJS and TypeScript

Advertising Sales:
Tammy Ferguson
832-717-4445 ext 26
tammy@codemag.com

This listing is provided as a courtesy to our readers and advertisers.
The publisher assumes no responsibility for errors or omissions.

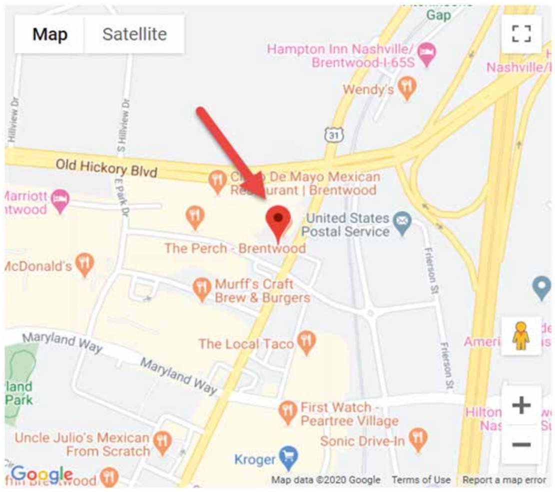


Figure 4: A marker helps identify your location on the map

Try It Out

Once you create your Google API key, substitute your key for the "YOUR_KEY_HERE" string included in the <script> tag from the last section. Run your page and if you've done everything correctly, you should see a map appear with your current location in the center of the map. If you don't see the map appear, check the browser's console window for any error messages that could guide you on how to fix the error.

Add a Marker on a Location

A popular item you want on a map is a large marker (see **Figure 4**) to denote the location you're trying to point out to the user. Google Maps makes it easy to add a marker on your map.

Instead of using your location to position the embedded Google Map, let's set up a fictitious company named "Paul's Training Company" located in Brentwood, TN, USA. Add two private variables within the closure.

```
let map = null;

let location = {
  "latitude": 36.036020,
  "longitude": -86.787600,
  "title": "Paul's Training Company",
  "marker": null
};
```

The *map* property is going to hold a reference to the Google Map object. The **location** property is a literal object with the **latitude**, **longitude**, and the **title** of the location. The **marker** property is currently set to null but is going to be set by code you write very soon.

Because you're not using the call to the geolocation object to get the coordinates, just call the addMap() function passing in the location variable. Modify the initialize() function in your closure to look like the following code snippet.

```
function initialize() {
  // Draw the map
  addMap(location);

  // Add marker for location
  addMarker(location);
}
```

Add an addMarker() function within the closure to create and add the marker at the coordinates in your location variable.

```
function addMarker(location) {
  // Create a new marker
  location.marker = new google.maps.Marker({
    position: new google.maps.LatLng(
       location.latitude, location.longitude),
    map: map,
    title: location.title
  });

  // Add marker to the map
  location.marker.setMap(map);
}
```

Try It Out

Save all your changes and run the page again. You should now see a marker on the embedded map. If you hover over the marker, you should see the value from the *title* property in your *location* property.

Add Multiple Locations

You may add as many markers as you want on your Google Map. You just need separate coordinates for each marker. Now you see the reason for creating a *location* literal object; it contains a property for each piece of data for each marker you wish to place on the map. Remove the private variable **location** you created earlier and replace it with an array named **locations**.

```
let locations = [];
```

Create a new function within the closure named createLocations() in which you push as many different literal objects into the locations array as you want to display on the map.

```
function createLocations() {
  locations.push({
    "latitude": 36.036020,
    "longitude": -86.787600,
    "title": "Paul's Training Company",
    "marker": null
  });
  locations.push({
    "latitude": 36.0339689,
    "longitude": -86.8058154,
    "title": "Mere Bulles",
    "marker": null
  });
}
```

You need to call this new function from the initialize() function, then loop through each item in the locations array and pass each item to the addMarker() method. Modify the initialize() function to look like the following.

```
function initialize() {
  // Create array of map definitions
  createLocations();
  // Draw the map
  addMap(locations[0]);

  // Draw the locations
  for (let index = 0; index <
       locations.length; index++) {
    // Add marker for location
    addMarker(locations[index]);
  }
}
```

Try It Out

Save your changes and try out this new code. You may have to scroll to the left to see the other marker depending on the size of your browser window. In the next section, you learn how to ensure the user can see all of your locations without having to scroll.

Center All Locations on Map

You shouldn't force your user to scroll to see all the locations you placed on your map. It would be nice if you could figure out the farthest latitude and longitudes to the north/south and east/west and somehow tell the map to automatically include these farthest points. Google Maps provides this in a **LatLngBounds** class. Create a **LatLngBounds** class as the first line of code in the initialize() function.

```
function initialize() {
  // Create a lat/lng boundary
  let bounds = new google.maps.LatLngBounds();

  // Rest of the code
}
```

A **LatLngBounds** class has an extend() method to which you pass in a **LatLng** class. The **LatLng** class contains two properties **lat** and **lng**. Your addMarker() function creates a **LatLng** object from the latitude and longitude in your map definition object. Call the extend() method after calling the addMarker() function within the loop in the initialize() function.

```
for (let index = 0; index <
         locations.length; index++) {
  // Add marker for location
  addMarker(locations[index]);

  // Extend boundaries to encompass marker
  bounds.extend(
    locations[index].marker.position);
}
```

After drawing all of the locations in the loop, call the fitBounds() method on the map passing in the **LatLngBounds** object. Add the following code immediately after the loop in the initialize() function.

```
// Fit the map to boundaries
if (locations.length > 1) {
  map.fitBounds(bounds);
}
```

Try It Out

Save your changes and run the page again. Now, the map should resize to the appropriate width to show you both locations you added.

Directions Between Two Points

Besides just showing markers on your map, you might want to provide your user with the ability to drive to your location from where they are. Google Maps provides the **DirectionsService** and the **DirectionsRenderer** classes to connect two points with a blue line, as shown in **Figure 5**.

Add a new function to your closure named addDirections(). This function creates an instance of a **google.maps.DirectionsService** class and an instance of **google.maps.DirectionsRenderer** class. Call the route() method on the **DirectionsService** class, passing in a literal object with the

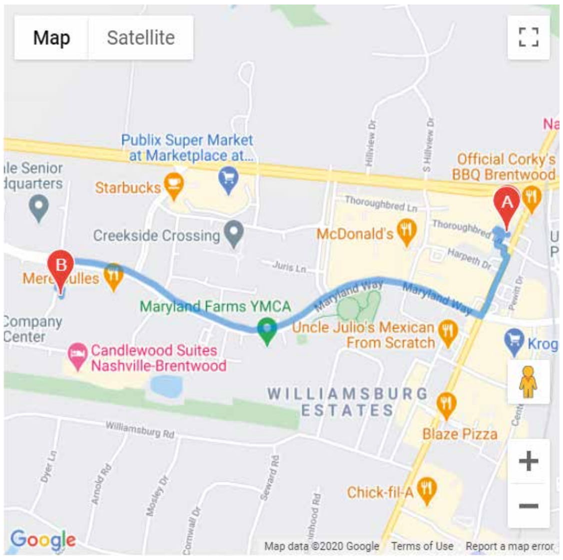
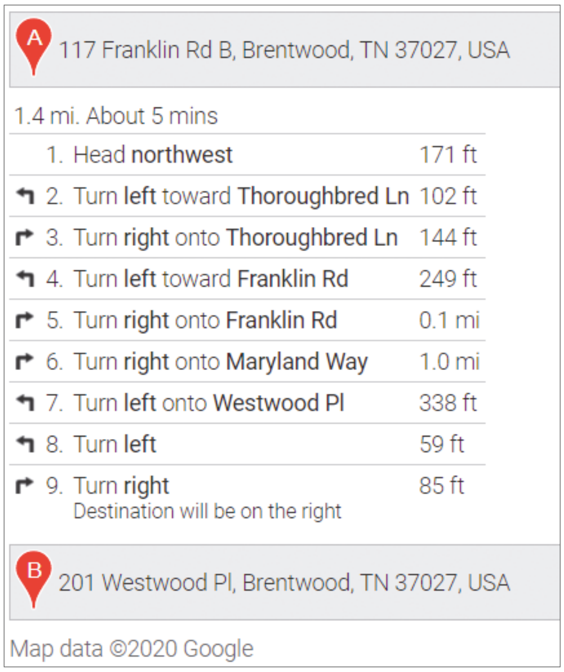


Figure 5: Google Maps can add a line showing how to travel from point A to point B.



A 117 Franklin Rd B, Brentwood, TN 37027, USA

1.4 mi. About 5 mins

1. Head **northwest**	171 ft
2. Turn **left** toward **Thoroughbred Ln**	102 ft
3. Turn **right** onto **Thoroughbred Ln**	144 ft
4. Turn **left** toward **Franklin Rd**	249 ft
5. Turn **right** onto **Franklin Rd**	0.1 mi
6. Turn **right** onto **Maryland Way**	1.0 mi
7. Turn **left** onto **Westwood Pl**	338 ft
8. Turn **left**	59 ft
9. Turn **right** Destination will be on the right	85 ft

B 201 Westwood Pl, Brentwood, TN 37027, USA

Map data ©2020 Google

Figure 6: Google Maps can add directions in text format between two points.

origin, **destination**, and **travelMode** properties set to the appropriate values. The second parameter to the route() method is a callback function in which you check the status to see if the directions were able to be rendered. If they are, call the setDirections() method passing in the response passed to the callback. Call the setMap() method and pass in the map object and Google Maps will draw the blue line as shown in **Figure 5**.

Call the new addDirections() function at the end of the initialize() function. Create two new **LatLng** objects from the first two entries in your **locations** array. For this article, you're using latitude and longitude to draw the directions between the two locations. However, the route() method on the **DirectionsService** class also accepts two addresses.

```
// Add directions between the two points
addDirections(
  new google.maps.LatLng(
    locations[0].latitude,
    locations[0].longitude),
  new google.maps.LatLng(
    locations[1].latitude,
    locations[1].longitude));
```

Try It Out
Save your changes to this page and redraw the Web page. You should now see a blue line connecting the first marker with the second marker. If you don't see a blue line, check the console window for any error messages.

Directions with Text
In addition to just showing a blue line, Google Maps can also display step-by-step directions as text on your Web page, as shown in **Figure 6**.

Determine where on your Web page you wish to have Google Maps draw the text directions and add a <div> in that location, as shown in the code snippet below.

```
<div id="directionsArea">
</div>
```

In the addDirections() function, add the code shown below after the call to the setMap() method. The code in bold calls the setPanel() method on the DirectionsRenderer class and you pass in a reference to the HTML <div> element within which you want Google Maps to render the text directions.

```
if (status === 'OK') {
  renderer.setDirections(response);
  // Render directions on the map
  renderer.setMap(map);

  // Call setPanel() to display text directions
  renderer.setPanel(
    document.getElementById("directionsArea"));
}
```

Try It Out
Save your changes to this page and redraw the Web page. You should now see a panel with text directions just below your map. If you don't see the text directions, check the console window for any error messages.

Information Windows
In **Figure 7**, you see a rectangular window above the marker with address information for the restaurant. This is called an information window and is accessed by the user clicking on the marker. It can be dismissed using the "x" in the upper right corner of the window.

There are just a few steps needed to add an information window to a marker. Create a variable, named **infoWindow**, in your closure and assign a null value to it for right now.

```
let infoWindow = null;
```

In the initialize() create a new instance a **google.maps.InfoWindow** class and assign it to the **infoWindow** variable.

```
function initialize() {
  // Create new InfoWindow if it is not already
  if (infoWindow == null) {
    infoWindow = new google.maps.InfoWindow();
  }

  // Rest of the code here
}
```

In the map definition literal object, add a new property named **infoContent**. In this variable is where you define the HTML and the text for what you want your information window to look like. Add this property and some different HTML for each marker in your *locations* array so you can see an information window for each marker.

```
locations.push({
  "latitude": 36.036020,
  "longitude": -86.787600,
  "title": "Paul's Training Company",
  "marker": null,
  "infoContent": "<div class='info-window'>
    <address>Paul's Training Company<br/>
      117 Franklin Rd<br/>
      Brentwood, TN 37027</address>
    </div>"
});
```

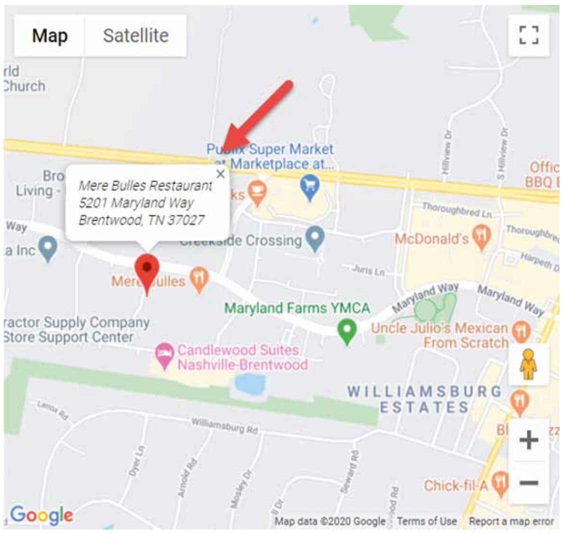

Mere Bulles Restaurant
5201 Maryland Way
Brentwood, TN 37027

Figure 7: Add an informational window to provide a better description of the location

Create an addInfoWindow() function in which you add an event listener to the marker you created earlier. On the click event of the marker call the setContent() method of the information window passing in the HTML you defined in the map definition object. Call the open() method on the **infoWindow** variable passing in the map object and the marker object. This is what displays the information window above the marker.

```
function addInfoWindow(location) {
  // Add click event to marker
  google.maps.event.addListener(location.marker,
    'click', function () {
    // Add HTML content for window
    infoWindow.setContent(location.infoContent);
    // Open the window
    infoWindow.open(map, location.marker);
  });
}
```

The addInfoWindow() function needs to be called after you have created your marker, as you are going to set up the click event on each marker you add to the map. Locate the loop in the initialize() function and add a call to the addInfoWindow() function after the call to the addMarker() function.

```
for (let index = 0; index <
     locations.length; index++) {
  // Add marker for location
  addMarker(locations[index]);

  // Add info window for location
  addInfoWindow(locations[index]);

  // Extend boundaries to encompass marker
  bounds.extend(locations[index]
    .marker.position);
}
```

Try It Out

Save your changes to this page and click on one of the locations. You should see an information window pop up above the marker. Now, click on another marker and the first information window disappears and one displays above the new marker you just clicked on. The reason for this toggling is that there's only one variable to hold an information window for the map object. If you change the variable, the original information window is destroyed and the new one appears.

Keep All Information Windows Open

Instead of only having a single information window appear at a time, you can keep them all open until the user manually closes each one. Remove the single instance of the **infoWindow** variable within your closure.

```
~~let infoWindow = null;~~
```

Add an **infoWindow** property to each **location** object and set the initial value to null.

```
locations.push({
  "latitude": 36.036020,
  "longitude": -86.787600,
  "marker": null,
  "infoWindow" : null,
  "title": "Paul's Training Company",
  "infoContent": "content here"
});
```

Remove the code that creates the instance of the single **infoWindow** variable from within the initialize() function.

```
function initialize() {
~~  // Create new InfoWindow if it is not already~~
~~  if (infoWindow == null) {~~
~~    infoWindow = new google.maps.InfoWindow();~~
~~  }~~

  // Rest of the code here
}
```

Locate the addInfoWindow() function and add a line of code to set the new **infoWindow** property on the *location* object to a new instance of a **google.maps.InfoWindow** class. Within the addListener() function be sure to call the setContent() and open() methods on the **infoWindow** property of the **location** object you pass in.

```
function addInfoWindow(location) {
  location.infoWindow =
    new google.maps.InfoWindow();

  // Add click event to marker
  google.maps.event.addListener(location.marker,
    'click', function () {
    // Add HTML content for window
    location.infoWindow.setContent(
      location.infoContent);
    // Open the window
    location.infoWindow.open(
      map, location.marker);
  });
}
```

Try It Out

Save your changes and when you now click on each marker, the previous information windows stay visible. You can choose which method you like better and use the appropriate code.

Summary

In this article, you learned to use the HTML 5 geolocation object to retrieve a user's current latitude and longitude. Most modern browsers support this object, so you shouldn't have any issues using this. The user must grant the usage of this object, so be sure to handle the case where your user doesn't want to give you their position. In addition to this object, you learned the basics of working with the Google Maps API. This API isn't free to use, but it's one of the best APIs available for mapping. You saw how to embed a map, display locations, and information windows using this API. There are many more APIs available, such as driving directions, geofencing, and much more. Take some time to explore the many different options.

Paul D. Sheriff

Getting the Sample Code

You can download the sample code for this article by visiting www.CODEMag.com under the issue and article, or by visiting www.pdsa.com/downloads. Select "Fairway/PDSA Articles" from the Category drop-down. Then select "Using Geolocation and Google Maps" from the Item drop-down.

ONLINE QUICK ID 2011041

Vue's Composition API

There was a bit of a controversy last year when the Vue team decided to show new ideas for composing components. Once the dust had settled, the Composition API was better for the battle. In this article, I'll show you how the Composition API works and why I think it's a better option for building your Vue applications going forward.

Shawn Wildermuth
shawn@wildermuth.com
wildermuth.com
twitter.com/shawnwildermut

Shawn Wildermuth has been tinkering with computers and software since he got a Vic-20 back in the early '80s. As a Microsoft MVP since 2003, he's also involved with Microsoft as an ASP.NET Insider and ClientDev Insider. He's the author of over twenty Pluralsight courses, written eight books, an international conference speaker, and one of the Wilder Minds. You can reach him at his blog at http://wildermuth.com. He's also making his first, feature-length documentary about software developers today called "Hello World: The Film." You can see more about it at http://helloworldfilm.com.

What's Wrong with the Options API?

First of all, nothing's wrong with the options API. If you're happy with the Options API, stay with it. But I hope you can see why the Composition API is a better way to do it. When I first started using Vue, I really liked how they used an anonymous object to set up a component. It was easy and felt natural. The longer I used it, the more it started to feel weird. Its reliance on the manipulating the **this** pointer was hiding some of the magic of how Vue works.

For example:

```
export default {
  data: () => {
    return {
      name: "Shawn",
    };
  },
  methods: {
    save: function () {
      alert(`Name: ${this.name}`); // MAGIC
    },
  },
  mounted: async function () {
    this.name = "Shawn's Name";    // MAGIC
  }
};
```

Because much of the work is done when the functions you define (e.g., save/mounted) are called, you have to really understand how the **this** pointer works. For example, can you tell me without compiling if this would have worked?

```
  mounted: async function () {
    this.load()
      .then(function() {
        this.name = "Shawn's Done";
      });
  }
```

No, it wouldn't. The reason is that the mounted function's **this** member has the Vue properties added. When you create a new nested function (e.g., in the call to **then()**), it creates a new scope and therefore a new **this** member. You'd have to remember to fix it by either using an Arrow function (e.g., **() =>**) or wrapping the this pointer. For example, this small fix works:

```
  mounted: async function () {
    this.load()
      .then(() => {  // The fix
        this.name = "Shawn's Done";
      });
  }
```

But that requires quite a lot of finesse to get right. What feels simple and right at first, turns into a bit of a headache.

In addition, the way that properties are made reactive is an additional piece of magic. When the members are returned from the data function, Vue wraps them (often using Proxy objects) to be able to know when the object changes. Of course, this can be confusing if you try to replace the object. For example:

```
export default {
  data: () => {
    return {
      name: "Shawn",
      items: []
    };
  },
  mounted: async function () {
    this.load()
      .then(result => {
        // breaks reactivity
        this.items = result.data;
      });
  }
};
```

This is one of the most common problems with Vue. Although these can be gotten around pretty simply, it's an impediment to learning how Vue works. Nothing is as frustrating as a bug that doesn't work and doesn't throw an error. These issues often end up happening with the Options API.

Lastly, there's a big issue with composing your components. Being able to use shared functionality was difficult with the Options API. Loading objects or methods that you then used in the component was hindered by the nature of the **Options** object. To combat that, Vue uses the notion of **mixins**. For example, you could create a mixin to extend a component like so:

```
import axios from "axios";

export default {
  data: function () {
    return {
      items: []
    };
  },

  methods: {
    load: async function() {
      let url =
        "https://restcountries.eu/rest/v2/all";
      let result = await axios.get(url)
      this.items.splice(0,
        this.items.length,
        ...result.data);
    },
    removeItem: function (item) {
      let index = this.items.indexOf(item);
      if (index > -1) {
        this.items.splice(index, 1);
      }
    }
  }
}
```

Note that the mixin looks a lot like the Vue Options API. It simply does a merge of the objects to allow you to add on to specific components. You use a mixin by specifying it in the component:

```
import dataService from "./dataService.mixin";

export default {
  mixins: [dataService],
  ...
```

The biggest issue with mixins is that you can't protect or easily know about name collision. It becomes trial and error to find out that a mixin was changed to use the same property or method of the component (or other mixins). Again, the magic is hidden behind the veneer of Vue and it makes debugging more difficult.

In the light of these limitations, the Composition API was born.

What's the Composition API?

Now that I've discussed the limitations of the Options API, let me introduce you to the Composition API. If you're still using Vue 2, you can still use the Composition API. To get started, you need to import the composition API library. First add it to your npm package:

```
> npm i @vue/composition-api --save
```

To enable the composition API, you simply have to register the composition API (usually in the main.js/ts):

```
import Vue from 'vue'
import VueCompositionAPI
       from '@vue/composition-api'

Vue.use(VueCompositionAPI)
```

Now that you've enabled it, let's take a look at the first Composition API component. At first, the look of a component looks similar:

```
export default {
  setup() {
    return {
    };
  },
};
```

It still starts with an anonymous object but the only member (so far) is a method called **setup**. Inside the **setup**, you'll return any data you need. This is very much like the function-type of data property in the Options API:

```
export default {
  setup() {

    const name = "Shawn";

    return {
      name
    };
  },
};
```

Instead of just assigning the value in the return, create it with a local variable inside the setup. Why would you? Because of JavaScript closures. Let's extend this and make a function that will be used in the markup:

```
export default {
  setup() {

    const name = "Shawn";

    function save() {
      alert(`Name: ${name}`);
    };

    return {
      name,
      save
    };
  },
};
```

The function can access the name because they're in the same scope. This is the magic of the Composition API. No magic, just JavaScript. What you can do with this pattern can become much more complex, but the basis of it all is just the closures. That's it.

In this example, the name is never changed. For Vue to be able to handle binding, it needs to know about any changes to the name. In this example, if you change the code to change the name in **save()**, it won't be reflected in the UI:

```
export default {
  setup() {

    const name = "Shawn";

    function save() {
      name = name + " saved"; // name changed
      alert(`Name: ${name}`);
    };

    return {
      name,
      save
    };
  },
};
```

Reactivity

When you return the object of the bindable object, Vue can see changes to the specific properties (the object is observed for changes). But because you're really assigning a new name in the Save method, there's no way for Vue to know it's been changed. It's still seeing the old name. To address this, Vue uses two methods of reactivity: ref and reactive wrappers.

To make the name reactive, you can just wrap it with a ref object:

```
import { ref } from "@vue/composition-api";
```

```
export default {
  setup() {

    const name = ref("Shawn");

    function save() {
      name.value = name.value + " saved";
      alert(`Name: ${name.value}`);
    };

    return {
      name,
      save
    };
  },
};
```

The name object is now wrapped with a **ref** object. This is a simple reactive that provides a property called **value** that represents the actual value. You're now assigning and reporting the **name** using the **value** property. Because it's using a ref wrapper, the changes will be notified to the user interface. This is an important concept, as you're surfacing some of the magic so that it's more obvious what's happening here.

This works well with primitive objects, but for objects with their own state (e.g., your classes or arrays), changes to the value aren't enough. What you really need is to use a proxy object so that any changes that happen inside functions on the object cause notifications of those changes.

```
import { ref, reactive }
       from "@vue/composition-api";

export default {
  setup() {

    const name = ref("Shawn");
    const items = reactive([]);

    function save () {
      // Change Array
      items.splice(0,items.length);
      name.value = name.value + " saved";
      alert(`Name: ${name.value}`);
    };

    return {
      name,
      save,
      items
    };
  },
};
```

In this example, because you're calling splice to change the item collection, Vue needs to know about this change. You do that by wrapping the complex object (the array, in this case) with another wrapper called **reactive()**.

The reactive wrapper from the Composition API is the same as Vue 2's Vue.observable wrapper.

One thing to note is that the object that's returned to bind to the UI is also reactive by the runtime:

```
    return {
      name,
      save,
      items
    }; // Object is reactive,
       // but the members aren't
       // automatically ref objects
```

Therefore, you don't need to wrap the object returning as a reactive object. Now that you have the basics of properties and reactivity, let's talk about how the Composition API allows you to compose your components differently.

Once you have ref and reactive objects, you can watch for changes. There are two methods to do this: **watch** and **watchEffect.** Watch allows you to respond to a change in a single object:

```
setup() {

  const name = ref("Shawn");

  watch(() => name,
        (before, after) => {
          console.log("name changes");
        });

    return {
      name
    };
  },
```

The **watch** function takes two parameters: a callback to return the data to **watch**, and a callback to be called when the change happens. It supplies the **before** and **after** values in case you need to use them.

Alternatively, you can use **watchEffect,** which watches for any changes in the reactive objects referred to in the component. It takes just a single callback:

```
  watchEffect(() => {
    console.log("Ch..ch...ch...changes.");
  });
```

Now you can use reactivity to not only make the markup react to changes, but also to have your own code react to those same changes.

Composing Components

When composing components, the Composition API simply encourages you to use the scope and closures to solve the problem. For example, you might create a simple factory to create the objects for your component:

```
import axios from "axios";

export default function () {

  const items = [];

  async function load() {
    const url =
      "https://restcountries.eu/rest/v2/all";
    let result = await axios.get(url);
    items.splice(0,
             items.length,
                 ...result.data);
  }

  function removeItem(item) {
    let index = items.indexOf(item);
    if (index > -1) {
      items.splice(index, 1);
    }
  }

  return {
    items,
    load,
    removeItem
  };
}
```

In this case, you create a function that generates the functionality you need. You can import it into your component and use it by calling the function:

```
import { ref,
  reactive,
  onMounted } from "@vue/composition-api";
import serviceFactory
       from "./dataService.factory";

export default {
  setup() {

    const name = ref("Shawn");

    // Create them by calling
    // the exported function
    const { load, removeItem, items} =
      serviceFactory();

    // Use Load from factory function
    onMounted(async () => await load());

    function save () {
      alert(`Name: ${this.name}`);
    };

    return {
      load,       // from factory function
      removeItem, // from factory function
      name,
      save,
      items       // from factory function
    };
  },
};
```

This pattern allows you to create factory functions to inject the functionality you need in your component. Remember, in this pattern, you're getting a new instance of these items on every call (which is often what you want). But if you wanted to share the data (like showing the same data on different forms), you could always use a simpler instance pattern:

```
import axios from "axios";

export let items = [];

export async function load() {
  const url =
    "https://restcountries.eu/rest/v2/all";
  let result = await axios.get(url);
  items.splice(0, items.length, ...result.data);
};

export function removeItem(item) {
  let index = items.indexOf(item);
  if (index > -1) {
    items.splice(index, 1);
  }
};
```

The difference here is that it's exporting each of the items separately (it doesn't generate them like in the factory pattern). If you use the items in separate components, you'll be manipulating the one instance. Using it is almost the same:

```
import { ref,
         reactive,
         onMounted }
        from "@vue/composition-api";

// Import them instead of generating them
import { load,
         items,
         removeItem } from "./dataService";

export default {
  setup() {

    const name = ref("Shawn");

    function save () {
      alert(`Name: ${this.name}`);
    };

    return {
      load,       // from import
      removeItem, // from import
      name,
      save,
      items       // from import
    };
  },
};
```

If you're already using Vuex, you can still use it in the Composition API. Vuex is more complex, but it does add some strictness to the read/write processes (only allowing changes in mutations). If you haven't worked with Vuex, I cover it in the January/February 2020 issue of CODE Magazine (https://www.codemag.com/Article/2001051/Vuex-State-Management-Simplified-in-Vue.js). Assuming you have a Vuex store created, the wiring up is different. There are no more helpers, as the magic that they do can be handled pretty simply:

```
import { ref,
         reactive,
         computed,
         onMounted }
        from "@vue/composition-api";
import store from "./store";

export default {
  setup() {
    const name = ref("Shawn");
    const items =
      computed(() => store.state.items);
    const removeItem = item => {
      store.commit("removeItem", item);
    };

    onMounted(async () => {
      await store.dispatch("load");
    );

    function save () {
      alert(`Name: ${this.name}`);
    };

    return {
      removeItem, // from function
      name,
      save,
      items       // from computed
    };
  },
};
```

Generally, state is turned into computed items: mutations and actions (e.g., **commit** and **dispatch**) are wrapped with simple functions. If necessary, they're returned in the object. Although the helpers were useful (and I'm sure someone will have a way to simplify this), this is, again, more obvious what's going on in these cases. You can compose your components more obviously using the Composition API. I've gotten to creating components, so let's talk about using components next.

Using Props

Because you're building components, you'd like to be able to use props to pass data to your components. How is this handled in the Composition API? Defining the props is the same as it is with the Options API:

```
export default {
  name: "WaitCursor",
  props: {
    message: {
      type: String,
      required: false
    },
    isBusy: {
      type: Boolean,
      required: false
    }

  },
  setup() {
  }
}
```

But because the properties aren't added to the magic **this** pointer, how do you use the properties in setup? You can get access to the properties by adding the optional parameter to setup:

```
  setup(props) {
    watch(() => props.isBusy,
          (b,a) => {
            console.log(`isBusy Changed`);
           });
  }
```

Additionally, you can add a second parameter to have access to the **emit**, **slots**, and **attrs** objects, which the Options API exposes on the **this** pointer:

```
  setup(props, context) {
    watch(() => props.isBusy,
          (b,a) => context.emit("busy-changed", a));
  }
```

Using Components

There are two ways to use components in Vue. You can globally register components (which is common for libraries) so that they can be used anywhere:

```
import WaitCursor
       from "./components/WaitCursor";

Vue.component("wait-cursor", WaitCursor);
```

More commonly, you'd add components to specific components that you're using. In Composition API, it's the same as it was with the Options API:

```
import WaitCursor
       from "./components/waitCursor";
import store from "./store";
import { computed } from "@vue/composition-api";

export default {
  components: {
    WaitCursor // Use Component
  },
  setup() {
    const isBusy =
      computed(() => store.state.isBusy);

    return {
      isBusy
    };
  },
};
```

Once you specify a component in the Composition API, your markup can just use it:

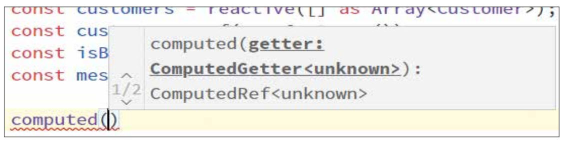
```
<div>
  <WaitCursor message="Loading..."
              :isBusy="isBusy"></WaitCursor>
  <div class="row">
    <div class="col">
      <App></App>
    </div>
  </div>
</div>
```

The way to use components isn't any different than it is in the Options API.

Using Composition API in Vue 3

If you're using Vue 3, you don't need to opt-into the Composition API. It's defaulted in the set up of a new project. The **@vue/composition-api** library is only used for Vue 2 projects. The real change for Vue 3 is that when you need to import Composition APIs, you need to get them directly from Vue:

```
import { ref,
         reactive,
         onMounted,
```



```
const customers = reactive([] as Array<Customer>);
const cus
const isB
const mes
computed()
```

```
computed(getter:
ComputedGetter<unknown>):
ComputedRef<unknown>
```

Figure 1: IntelliSense in TypeScript

```
        watch,
        watchEffect }
  //from "@vue/composition-api";
  from "vue";
```

Everything else is just the same. Just import from "vue". In Vue 3, it's just a little simpler to use the Composition API as it's the default behavior.

Using Composition API in TypeScript and Vue 3

One of the main goals of Vue 3 was to improve the TypeScript experience. Using the Composition API certainly benefits from using TypeScript. There are type libraries for all of what I've talked about. But to add type safety, you do have to make some small changes to use TypeScript. First, when you create a component, you have to wrap your component's object with **defineComponent**:

```
import {
  defineComponent,
  reactive,
  onMounted,
  ref
} from "vue";

import Customer from "@/models/Customer";

export default defineComponent({
  name: "App",
  setup() {
    ...
  }
});
```

Additionally, you can use types in your set up:

```
    const customers =
      reactive([] as Array<Customer>);
    const customer = ref(new Customer());
    const isBusy = ref(false);
    const message = ref("");
```

In these examples, the variables are inferred as types (for example, the Customers object is Reactive<Customer[]>). In addition, the API is typed so it will lower your chances to pass in the wrong data. Of course, additionally, IntelliSense is a big benefit if you're using TypeScript (especially in Visual Studio or VS Code), as seen in **Figure 1**.

Where Are We?

Whether you're a veteran Vue developer or brand new, getting used to the Composition API is going to require you to change your mindset. The move from convention to something that's more explicit may be uncomfortable, but I've found that I like the new model a lot better. I spend less time scratching my head about what's happening with function scope and trying to remember when I can and can't use the arrow functions. I hope I've convinced you.

Shawn Wildermuth

CODE

SPONSORED SIDEBAR:

Need FREE Project Advice? CODE Can Help!

No strings free advice on a new or existing software development projects. CODE Consulting experts have experience in cloud, Web, desktop, mobile, microservices, and DevOps projects. Schedule your free Hour of CODE call with our expert consultants (not a sales call!). For more information, visit www.codemag.com/consulting or email us at info@codemag.com.

dtSearch®

Instantly Search Terabytes

dtSearch's **document filters** support:
- popular file types
- emails with multilevel attachments
- a wide variety of databases
- web data

Over 25 search options including:
- efficient multithreaded search
- easy multicolor hit-highlighting
- forensics options like credit card search

Developers:
- SDKs for Windows, Linux, macOS
- Cross-platform APIs for C++, Java and .NET with .NET Standard / .NET Core
- FAQs on faceted search, granular data classification, Azure and more

Visit dtSearch.com for
- hundreds of reviews and case studies
- fully-functional enterprise and developer evaluations

The Smart Choice for Text Retrieval® since 1991

1-800-IT-FINDS
www.dtSearch.com

Introduction to the Go Programming Language

Go (aka Golang) is one of the fastest growing programming languages. It's an open-source language released by Google in 2009 and created by Ken Thompson (designer and creator of UNIX and C), Rob Pike (co-creator of UTF 8 and UNIX format), and Robert Griesemer. It's a multi-purpose programming language specifically designed to build scalable and faster applications.

Wei-Meng Lee
weimenglee@learn2develop.net
http://www.learn2develop.net
@weimenglee

Wei-Meng Lee is a technologist and founder of Developer Learning Solutions (www.learn2develop.net), a technology company specializing in hands-on training on the latest technologies. Wei-Meng has many years of training experiences and his training courses place special emphasis on the learning-by-doing approach. His hands-on approach to learning programming makes understanding the subject much easier than reading books, tutorials, and documentation. His name regularly appears in online and print publications such as DevX.com, MobiForge.com, and CODE Magazine.

Although Go has been around for quite a while now, it didn't manage to get wide adoption by developers until more recently due to the proliferation of cloud computing and microservices. Today, Go has been widely used by major companies such as Google, Dropbox, Uber, and Dailymotion.

In this article, I'll walk you through the language and dive into some areas where Go shines. By the end of this article, you should have a pretty solid feel of Go and be on your way to writing some cool Go packages.

Getting Started with Go

Installing Go on your computer is straight-forward—go to https://golang.org/dl/ and download the installer for the OS you are using (see **Figure 1**).

You can use your favorite code editor to write Go code. I use Visual Studio Code.

Hello World!

In the spirit of adhering to tradition, let's create a text file named **helloworld.go** and populate it with the following statements:

```
package main

import "fmt"

func main() {
    fmt.Println("Hello, world!")
}
```

The first line indicates the name of this package, which is **main**. Packages are used in Go to organize and reuse code. Within this **main** package, you have the **main()** function, which is the function to call when you start your program. Note that you also import another package called **fmt**, which is a package that implements formatted I/O. This package contains functions (e.g., **Println**) that allow you to print output to the console, similar to C's **printf()** and **scanf()** functions.

To run the **helloworld.go**, you can first build the program using the **go** tool with the **build** command:

```
$ go build helloworld.go
```

A binary will then be created. You can now run the binary and see the output:

```
$ ./helloworld
Hello, world!
```

Alternatively, you can also build and run the program using the **run** command:

```
$ go run helloworld.go
Hello, world!
```

You can also use the built-in println() and print() function for printing purposes.

Variables

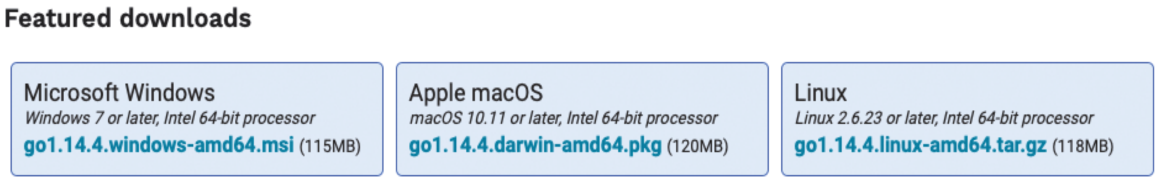
There are a couple of ways to declare variables in Go. To declare a variable explicitly, you use the **var** keyword:

```
var num1 = 5               // type inferred
var num2 int = 6           // explicitly typed
var rates float32 = 4.5   // declare as float32
                           // and initialize

var raining bool = false // declare as bool
                           // and initialize
```

Notice that you can either explicitly specify the type of variable, or let the compiler infer it for you. When you are declaring a variable and initializing it, you should use type inference. Otherwise, you need to specify the type explicitly:

```
var str string             // declare as string
```

Variables declared without initialization are zero-valued. For example, **str** above would have an initial value of "" and an integer variable has a value of 0.



Featured downloads

Microsoft Windows
Windows 7 or later, Intel 64-bit processor
go1.14.4.windows-amd64.msi (115MB)

Apple macOS
macOS 10.11 or later, Intel 64-bit processor
go1.14.4.darwin-amd64.pkg (120MB)

Linux
Linux 2.6.23 or later, Intel 64-bit processor
go1.14.4.linux-amd64.tar.gz (118MB)

Figure 1: Downloading the Go installer for your OS

There's a shortcut for declaring and initializing variables without needing to use the **var** keyword. This is done using the **:=** operator, like the following:

```
num3 := 7               // declare and init
num4 := num3
```

You can also declare multiple variables and assign them in a single statement, like this:

```
var num5, num6 int = 8, 9  // multiple
                           // declares and
                           // assignment
```

Here's another example where you can declare and initialize multiple variables:

```
var (
    age = 25
    name = "Samuel"
)
```

String Interpolation

One of the common things you do in programming is printing out the values of variables in a string. Consider the following declarations:

```
    var num1 = 5
    var rates float32 = 4.5
```

Suppose you want to print out the values of these two variables in a string. To do that, you need to convert the two numeric variables using the **strconv** package:

```
import (
    "fmt"
    "strconv"
)
...
    fmt.Println("num1 is " +
        strconv.Itoa(num1) +
        " and rates is " +
        strconv.FormatFloat(
        float64(rates),'f',2,32))
    // output is:
    // num1 is 5 and rates is 4.50
```

The **strconv** package contains a number of functions for converting numeric/Boolean values to strings, such as:

```
s := strconv.FormatBool(true)
s := strconv.FormatFloat(3.1415, 'E', -1, 64)
s := strconv.FormatInt(-42, 16)
s := strconv.FormatUint(42, 16)
```

There are also functions to convert strings to numeric, such as:

```
b, err := strconv.ParseBool("true")
f, err := strconv.ParseFloat("3.1415", 64)
i, err := strconv.ParseInt("-42", 10, 64)
u, err := strconv.ParseUint("42", 10, 64)
```

An easy way to combine string and numeric values is to use the **Sprintf()** function with the various format specifiers, like the following:

```
str := fmt.Sprintf(
    "num1 is %d and rates is %.2f",
    num1, rates)
fmt.Println(str)
```

Data Structures

Go supports a number of data structures:

- Arrays
- Slices
- Maps
- Struct

The following sections discuss each of these in more detail.

Arrays

In Go, an array has fixed size. That is, once an array is declared, its size cannot be changed. The following shows some examples of declaring arrays of specific sizes:

```
var nums [5] int     // int array of 5 items
fmt.Println(nums)    // [0 0 0 0 0]

var names [3] string // string array of 3
                     // items
fmt.Println(names)   // [  ]

var ended [3] bool   // bool array of 3 items
fmt.Println(ended)   // [false false false]
```

Array elements are zero-based, and you can access them individually and also assign values to them:

```
names[0] = "iOS"
names[1] = "Android"
names[2] = "Symbian"
fmt.Println(names)   // [iOS Android Symbian]
```

Slices

As mentioned, arrays in Go are fixed in size. A **Slice** in Go is a light-weight data structure that's more flexible than arrays. Think of slices as a view into an array.

Let's see how slices are created:

```
x := make([] int, 5)  // creates a slice of 5
                      // elements, capacity =
                      // 5
fmt.Println(x)        // [0 0 0 0 0]
```

The **make()** function allocates and initializes an array of the specified type. In the above code snippet, **x** is a slice of five elements. You can also create a slice of two elements, but with a maximum capacity of three:

```
x = make([] int, 2, 3)  // creates a slice
                        // of 2 elements,
                        // capacity = 3
fmt.Println(x)          // [0 0]
```

In the above example, **x** now has two elements, but it can contain a maximum of three items. An easier way to write a slice is this:

```
odds := [] int {1,3,5}
fmt.Println(odds)    // [1 3 5]
```

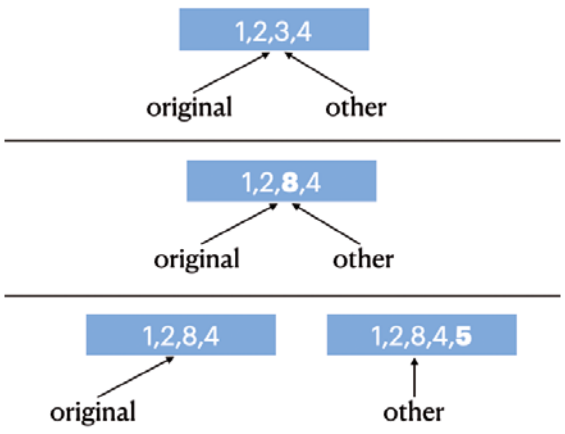


Figure 2: The original and other slices

In fact, if you recall, earlier you declared an array using this:

```
var nums [5] int    // nums is an array
```

If you remove the 5, **nums** is now a slice and not an array:

```
var nums [] int     // nums is now a slice
```

Understanding the Behavior of Slices

Consider the following code snippet:

```
original := []int{1,2,3,4}
other := original
```

In the above code snippet, **original** is a slice of capacity four. After you assigned **original** to **other**, **other** is now a reference to **original** (see the top of **Figure 2**).

Now, when you make changes to the third element in **other** like this:

```
other[2] = 8
```

Both slices now print the same values (see also the middle of **Figure 2**):

```
fmt.Println(original)       // [1 2 8 4]
fmt.Println(other)          // [1 2 8 4]
```

If you append an item to **original** and then assign it to **other**:

```
other = append(original, 5)
```

Then **other** now points to a new slice (as it has exceeded its capacity of four), as shown in the bottom of **Figure 2**. So when you now make changes to **other**, **original** won't be affected:

```
other[2] = 9
fmt.Println(other)          // [1 2 9 4 5]
fmt.Println(original)       // [1 2 8 4]
```

Consider another example, where you now have a slice of two elements but with a capacity of four:

```
x := make([] int, 2, 4)
fmt.Println(x)              // [0 0]
```

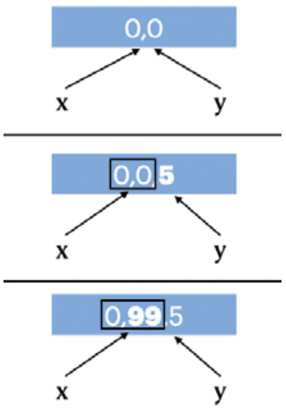


Figure 3: The x and y slices

Let's now assign **x** to **y** (see the top of **Figure 3**):

```
y := x
fmt.Println(x)              // [0 0]
fmt.Println(y)              // [0 0]
```

If you now append an item to **x** and then assign it back to **y**:

```
y = append(x,5)
fmt.Println(x)           // [0 0]
fmt.Println(y)           // [0 0 5]
```

Then **x** still points to the original two numbers and **y** now points to the same numbers, plus the additional one appended to **x** (see the middle of **Figure 3**). This is because **y** (as well as **x**) has the capacity of four and has room for up to four items.

When you now modify the second item in **y**, both **x** and **y** are affected (see the bottom of **Figure 3**):

```
y[1] = 99
fmt.Println(x)           // [0 99]
fmt.Println(y)           // [0 99 5]
```

Slicing on Slices/Arrays

You can perform slicing (extracting a range of values) on arrays and slices. Consider the following array:

```
var c[3] string
c[0] = "iOS"
c[1] = "Android"
c[2] = "Windows"
```

To extract the first two items, you can use the following slicing:

```
b := c[0:2]
fmt.Println(b)      // [iOS Android]
```

The result of the slicing (**b**) is a slice. You can print the capacity of **b** using the **cap()** function:

```
fmt.Println(cap(b)) // 3
```

Observe that the capacity of **b** assumes the capacity of the underlying array - **c.** You can change the capacity of the slice **b**, by specifying the capacity as the third argument in the slicing:

```
b = c[0:2:2]
fmt.Println(b)      // [iOS Android]
fmt.Println(cap(b)) // 2
```

Maps

Besides array, another essential data structure is a dictionary. In Go, this is known as a **map**, which implements a hash table. The following statement declares a map type called **heights**:

```
var heights map[string] int
```

The following statement initializes the map using the **make()** function:

```
heights = make(map [string] int)
```

The following statement declares and initializes an empty map:

```
weights := map[string] float32 {}
```

You can also declare and initialize the map variable with some values:

```
weights := map[string] float32 {
    "Peter": 45.9,
    "Joan": 56.8,
}
```

The following statement adds a new key/value pair to the **heights** map:

```
heights["Peter"] = 178
```

To delete the key/value pair, use the **delete()** function:

```
delete(heights, "Peter")
```

To check whether a key exists in the map, use the following code snippet:

```
if value, ok := heights["Peter"]; ok {
    fmt.Println(value)
} else {
    fmt.Println("Key does not exists")
}
```

If the key exists, the **ok** variable will be set to **true**; otherwise, it will be set to **false**. You can also iterate through a map using the **for** loop together with the **range** keyword:

```
// iterating over a map
for k, v := range heights {
    fmt.Println(k,v)
}
```

Structs

Go doesn't have classes, but it supports structs. The following shows the **Point** struct containing two members:

```
type Point struct {
    X float64
    Y float64
}
```

You can also define methods on structs. A method is a function with a special receiver argument. To add a method to a struct, define a function with the struct passed in as an argument defined before the function name, like this:

```
func (p Point) Length() float64 {
   return math.Sqrt(math.Pow(p.X,2.0) +
          math.Pow(p.Y,2.0))
}
```

The following statement creates an instance of the **Point** struct:

```
ptA := Point{5,6}
```

If you want to create a reference to another struct, use the **&** character:

```
ptB := &ptA       // assigning a reference
```

Here, **ptB** is a reference to **ptA**. To prove this, modify the value of **X** through **ptB** and then print out the values of **ptA** and **ptB**:

```
ptB.X = 55
fmt.Println(ptA)       // {55 6}
fmt.Println(ptB)       // &{55 6}
```

You can call the **Length()** method of the **Point** struct like this:

```
fmt.Println(ptA.Length())   // 7.810...
```

Here is another example of creating a new instance of the **Point** struct:

```
pt1 := Point{X:2,Y:3}
pt2 := pt1             // making a copy
```

Now **pt2** is a copy of **pt1**. As usual, the following statements prove this:

```
pt2.X = 22
fmt.Println(pt1)       // {2 3}
fmt.Println(pt2)       // {22 3}
```

Decision-Making and Looping Constructs

Go's decision-making statements are very similar to other languages. It supports the standard **if-else** statement and **switch** statement, but surprisingly, no ternary statement. For looping, there's only one looping construct: the **for** loop.

The following sections will discuss these in more detail.

If-else

Decision making in Go is very similar to other languages:

```
if true {
    fmt.Println(true)
} else {
    fmt.Println(false)
}
```

Interestingly, there's no ternary operator in Go. However, the **if** statement allows you to have two expressions in it: one assignment and one condition. Consider the following:

```
limit := 10
if sum := addNums(5,6); sum <= limit {
    fmt.Println(sum)
} else {
    fmt.Println(limit)
}
// prints out 10
```

In the above, the **if** statement first evaluates the **addNums()** function and assigns the result to **sum**. It then evaluates the condition to check if **sum** is less than or equal to **limit**.

Switch Statements

If you need to evaluate multiple conditions, use the **switch** statement:

```
grade := "B"
switch grade {
case "A":
    fallthrough
case "B":
    fallthrough
case "C":
    fallthrough
case "D":
    fmt.Println("Passed")
case "F":
    fmt.Println("Failed")
default:
    fmt.Println("Undefined")
}
// Passed
```

There's no need to specify the break statement in a **switch** statement in Go. Once a condition is matched and its associated block evaluated, it breaks automatically from the **switch** statement. If you want to have the default behavior in C, use the **fallthrough** keyword.

Looping

Similar to most languages, Go has the **for** looping construct:

```
for i:=0; i<5; i++ {
    fmt.Println(i)
}
```

You can use the **for** loop to run an infinite loop, like this:

```
for {
}
```

There is no **while** loop in Go, because you can improvise it using the **for** loop:

```
counter := 0
for counter <5 {
    fmt.Println(counter)
    counter++
}
```

You can use the **continue** statement to force the **for** loop to continue with the next iteration of the loop, skipping all the code thereafter:

```
// prints 0 to 9 except 5
for i:=0; i<10; i++ {
    if i==5 {
        continue
    }
    fmt.Println(i)
}
```

The **break** statement, on the other hand, exits a for **loop** prematurely:

```
// prints 0 to 4
for i:=0; i<10; i++ {
    if i==5 {
        break
    }
    fmt.Println(i)
}
```

Ranging

To iterate over an array or slice, you use the **range** keyword. When used with the **for** loop construct the **range** keyword returns an index and item of each element in the array/slice. Here's an example:

```
    primes := [] int {2, 3, 5, 7, 11, 13}
    for i, v := range primes {
        fmt.Println(i, v)
    }
```

The above code snippet prints out the following:

```
0 2
1 3
2 5
3 7
4 11
5 13
```

You can also iterate through a string using the **range** keyword and the **for** loop:

```
s:= "Hello, world!"
for _, c := range s {
    fmt.Printf("%c\n", c)
}
```

When you iterate through a string, it returns the ASCII code for each character in the string. To print it out as a character, you need to use the **Printf()** function with the %c format specifier.

Functions

In Go, you define a function using the **func** keyword:

```
func doSomething() {
    fmt.Println("Hello")
}

func main() {
    // calling a function
    doSomething()
}
```

If the function returns a value, you specify the return value type at the end of the function name:

```
// returns int result
func addNum(num1 int, num2 int) int {
    return num1 + num2
}
```

Multiple Return Values

Functions can also return multiple values, very much like tuples in some languages (like Python):

```
func countOddEven(s string) (int,int) {
    odds, evens := 0, 0
    for _, c := range s {
        if int(c) % 2 == 0 {
            evens++
        } else {
            odds++
        }
    }
    return odds,evens
}
```

```
odds, evens := countOddEven("123456789")
```

The above **countOddEven()** function can also be rewritten using **named return types**:

```
func countOddEven(s string) (odds,evens int) {
    ...
    return
}
```

Variadic Functions

Go supports **variadic functions**, which are functions with a variable number of arguments:

```
func addNums(nums ... int) int {
    total := 0
    for _, n := range nums {
        total += n
    }
    return total
}
```

To call the **addNums()** function, you can now pass in any number of arguments:

```
    sums := addNums(1,2,3)
    fmt.Println(sums) // 6

    sums = addNums(1,2,3,4,5,6)
    fmt.Println(sums) // 21
```

Anonymous Functions

An anonymous function is a function without a name. Consider the following statement:

```
var i func() int
```

Here, **i** is declared to be a function that returns **int** value. You can now provide an implementation for **i**:

```
i = func() int {
    return 5
}
```

To invoke the anonymous function, you call **i** the way you call a normal function, like this:

```
fmt.Println(i())      // 5
```

Closures

Anonymous functions are very useful when implementing closures. A closure is a function that references variables from outside its body. Closures allow you to pass in functions as arguments into functions. To understand closure, it's useful to see a concrete example.

Most programming languages that support closures (AKA lambda functions) come with the predefined **filter()**, **map()**, and **reduce()** functions. However, Go doesn't come with these predefined functions. So let's now implement the **filter()** function in Go using closures. Consider the following **filter()** function:

```
func filter(arr [] int,
    cond func(int) bool) [] int {
    result := [] int{}
    for _,v := range arr {
        if cond(v) {
            result = append(result, v)
        }
    }
    return result
}
```

It takes in two arguments: an **int** array and an anonymous function (**cond**), which itself takes in an **int** value and returns a **bool** result. Within this **filter()** function, you iterate through each of the items in the **arr** array, and call the **cond** anonymous function. If the **cond** anonymous function evaluates to **true**, the item in the array is appended to the **result** array.

Now if you have an array and want to extract all even numbers from the array, you can call the **filter()** function and write your own filtering logic using the anonymous function:

```
    a := [] int {1,2,3,4,5}
    fmt.Println(
        filter(a,
            func(val int) bool {
                return val%2==0
            }))
```

To extract those numbers that are multiple of threes, you can simply modify the expression inside the anonymous function:

```
    a := [] int {1,2,3,4,5}
    fmt.Println(
        filter(a,
            func(val int) bool {
                return val%3==0
            }))
```

Goroutines

Most developers are familiar with threading. Threading allows you to implement concurrent operations: multiple functions all running at the same time. In Go, a **goroutine** is a light-weight thread managed by the Go runtime. To run a function as a goroutine, simply call it using the **go** keyword.

Consider the following example:

```
package main

import (
    "fmt"
    "time"
)

func say(s string, times int) {
    for i := 0; i < times; i++ {
        time.Sleep(100 * time.Millisecond)
        fmt.Println(i, s)
    }
}

func main() {
    go say("Hello", 3)
    go say("World", 2)

    // prevent main() from exiting
    fmt.Scanln()
}
```

In the above code snippet, you have a function called **say()**. It takes in a string and a number. The number indicates how many times the given string is to be printed on the console. There's a delay of 100ms between each printing. In the **main()** function, you call the **say()** function twice, each one with the **go** keyword:

```
    go say("Hello", 3)
    go say("World", 2)
```

The first statement calls the **say()** function as a goroutine. Essentially, it means "go and run the **say()** function independently and immediately return control back to the calling statement." The second statement does the same. Now you have two separate instances of the **say()** function running concurrently. The result may appear like this (you may get a different result):

```
0 World
0 Hello
1 World
1 Hello
2 Hello
```

Each time you run this, you might get a slightly different sequence of the words printed. This is because the Go runtime manages how this functions runs, and you have no control over which is printed first. Observe that the **main()** function has the following statement:

```
fmt.Scanln()
```

Without this statement, you'd most likely be unable to see any outputs. This is because each time a goroutine is called, the control is immediately returned back to the calling statement. Without the **Scanln()** function to wait for user input, the program automatically terminates after the second goroutine is called. Once the program is terminated, all goroutines are also terminated and no output will ever be printed.

Channels

Goroutines are executed independently of one another. But they can communicate with one another through pipes known as ***channels***. In Go, channels are the pipes that connect concurrent goroutines. You can send values into channels from one goroutine and receive those values in another goroutine. To understand the usefulness of channels, consider the following example. Suppose you have a function named **sum()** that sums up an array of integer values:

```
func sum(s []int, c chan int) {
    sum := 0
    for _, v := range s {
        sum += v
    }
    c <- sum
}
```

Notice that the function has a second parameter:

```
func sum(s []int, c chan int) {
```

The **chan** keyword represents a channel, and in this example, it's a channel of type **int**. When the numbers in the array have been summed up, the sum is written to the channel via this syntax:

```
c <- sum
```

To use the **sum()** function, let's now generate 10 random numbers and assign it to an array, **s**:

```
    rand.Seed(time.Now().UnixNano())
    s := []int {}
    for i := 0; i < 10; i++ {
        s = append(s, rand.Intn(100))
    }
```

Let's also create a channel to store **int** values:

```
c := make(chan int)
```

Although we only have 10 items in the array, imagine if you have 1 million items. It will take some time to sum up all the numbers in the array. For this example, you'll split this array into five parts, take each part and pass it to the **sum()** function together with the channel **c**, and call it a goroutine:

```
parts := 5
partSize := 2
i := 0
for i<parts {
    go sum(s[i*partSize:(i+1)*partSize], c)
    i += 1
}
```

Essentially, you're breaking up the array into five parts and trying to sum each part concurrently. As each goroutine finishes the summing process, it writes the partial sum to the channel, as shown in **Figure 4**.

Channels behave like queues: All items are retrieved in the same order that they were written (First-In-First-Out).

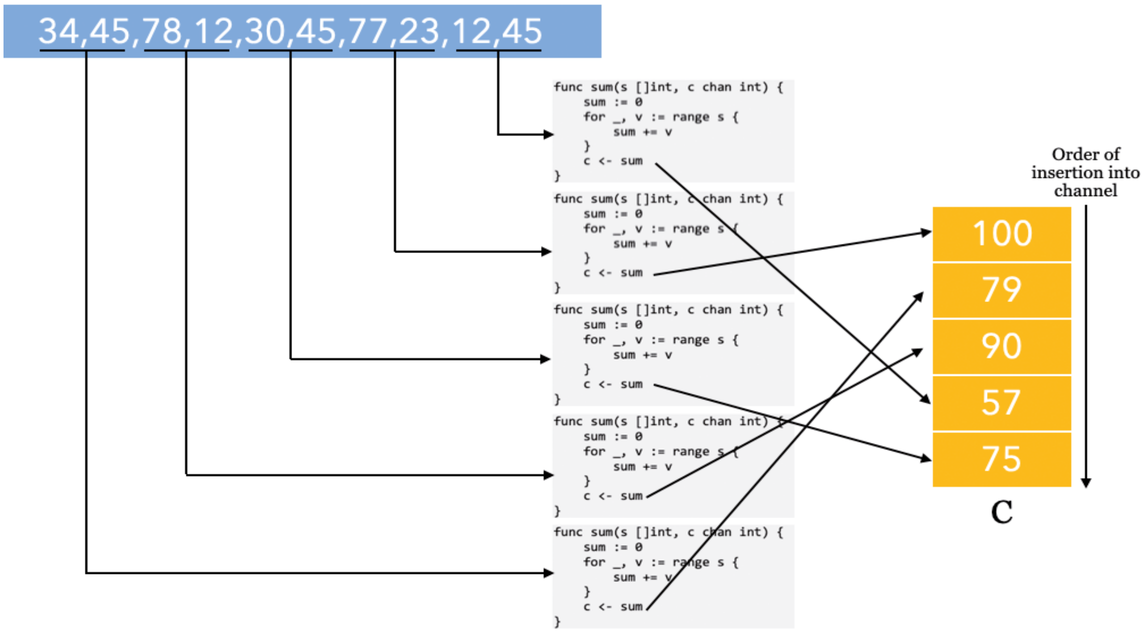
Because you know that you have five separate goroutines (and therefore five values to be written to the channel), you can write a loop and try to extract the five values in the channel:

```
i = 0
total := 0
for i<parts {
   partialSum := <-c    // read from channel
   fmt.Println("Partial Sum: ", partialSum)
   total += partialSum
   i += 1
}
fmt.Println("Total: " , total)
```

Each value in the channel represents the partial sum of the values in each array. It's important to know that when you send a value into a channel, the goroutine is blocked until the value is received by another function/goroutine. Likewise, when you're reading a value from a channel, your code is blocked until the data is read from the channel. In the event that the goroutines are taking a long time to sum up, the above code snippet will block until all the partial sums are retrieved. **Listing 1** shows a complete program where you can simulate the **sum()** function summing up 1000 numbers.

Go Packages and Modules

Go uses the concept of packages to better organize code for reusability and readability. So far, you've seen how to use some of the built-in packages like **fmt**, **strconv**, **math**, and **time** in your Go application. In this section, you'll dive into



```
func sum(s []int, c chan int) {
    sum := 0
    for _, v := range s {
        sum += v
    }
    c <- sum
}
func sum(s []int, c chan int) {
    sum := 0
    for _, v := range s {
        sum += v
    }
    c <- sum
}
func sum(s []int, c chan int) {
    sum := 0
    for _, v := range s {
        sum += v
    }
    c <- sum
}
func sum(s []int, c chan int) {
    sum := 0
    for _, v := range s {
        sum += v
    }
    c <- sum
}
func sum(s []int, c chan int) {
    sum := 0
    for _, v := range s {
        sum += v
    }
    c <- sum
}
```

Figure 4: Goroutines adding values to a channel

Listing 1: Demonstration of the use of channels

```
package main

import (
    "fmt"
    "math/rand"
    "time"
)

func sum(s []int, c chan int) {
    sum := 0
    for _, v := range s {
        sum += v
        time.Sleep(time.Duration(rand.Intn(100)) *
            time.Millisecond)
    }
    c <- sum
}

func main() {
    rand.Seed(time.Now().UnixNano())
    s := []int {}
    for i := 0; i < 1000; i++ {
        s = append(s, rand.Intn(100))
    }
    fmt.Println(s)

    c := make(chan int)
    parts := 5
    partSize := 200
    i := 0
    for i<parts {
        go sum(s[i*partSize:(i+1)*partSize], c)
        i += 1
    }

    i = 0
    total := 0
    for i<parts {
       partialSum := <-c
       fmt.Println("Partial Sum: ", partialSum)
       total += partialSum
       i += 1
    }
    fmt.Println("Total: " , total)
}
```

the topic of packages and modules in more detail. You will also learn how to create your own packages and make them available to fellow developers for use.

Go Packages

So far, you've seen that your Go applications always have this first statement:

```
package main
```

Go organizes code into units called **packages**. A package is made up of a collection of files. The **main** package is a special package that contains the **main()** function, and this makes the main package an executable program. The **main()** function serves as the entry point to your application. All files in a package must be in the same directory and all package names must be in all lowercase.

Let's take a look at one example. Suppose you have a directory named **my_app** and in it is a file named **helloworld.go**:

```
$HOME
  |__my_app
    |__helloworld.go
```

The content of the **helloworld.go** file looks like this:

```
package main

import (
    "fmt"
    "math"
)

type Point struct {
    X float64
    Y float64
}

func (p Point) Length() float64 {
    return math.Sqrt(math.Pow(p.X,2.0) +
           math.Pow(p.Y,2.0))
}

func main() {
    pt1 := Point{X:2,Y:3}
    fmt.Println(pt1)
}
```

Observe that the package is named **main** and so it has the **main()** function. You can extract the definition of the **Point** struct as well as its method **Length()** to another file, say, **point.go**, and put it in the same directory as **helloworld.go**:

```
$HOME
  |__my_app
    |__helloworld.go
    |__point.go
```

The content of **point.go** looks like this:

```
package main

import (
    "math"
)

type Point struct {
    X float64
    Y float64
}

func (p Point) Length() float64 {
    return math.Sqrt(math.Pow(p.X,2.0) +
           math.Pow(p.Y,2.0))
}
```

It's important to make sure that the first line uses the same **main** package name. With the **Point** struct and the **Length()** method removed, **helloworld.go** now looks like this:

```
package main

import (
    "fmt"
)

func main() {
    pt1 := Point{X:2,Y:3}
    fmt.Println(pt1)
}
```

Because these two files—**helloworld.go** and **point.go**—all reside in the same directory and they have the same package name (**main**), they are deemed to be of the same package. To run the above application, type the following commands in Terminal:

```
$ cd ~/my_app
$ go run *.go
{2 3}
```

For this to work, you need to ensure that:

- Both files are in the same directory
- Both packages have the same package name (**main**)
- One of the files has a **main()** function

Using Third-Party Packages

Unlike languages likes Python or JavaScript where you can download third-party packages from central repositories like PyPI or NPM, Go doesn't have a centralized official package registry. Instead, you simply fetch third-party packages through a hostname and path. For example, there's a Go package located at https://github.com/hackebrot/turtle that allows you to obtain emojis based on names. To install that package, you simply use the **go get** command followed by the URL of the package (without the "https://"), like this:

```
$ go get github.com/hackebrot/turtle
```

Once you do that, the **github.com/hackebrot/turtle** package is installed in the **~/go/src** folder of your local computer:

```
$HOME
  |__go
    |__src
    |  |__github.com
    |    |__hackebrot
    |    |  |__turtle
    |    |    |__ ...
    |    |    |__ ...
```

To use the package, you simply import it into your package, like this:

```
package main

import (
    "fmt"
    "github.com/hackebrot/turtle"
)

func main() {
    emoji, ok := turtle.Emojis["smiley"] //😃
    if !ok {
        fmt.Println("No emoji found.")
    } else {
        fmt.Println(emoji.Char)
    }
}
```

Creating Go Modules

So far, the package you created in the previous section can be run directly as an executable program. However, a package is more useful if it contains functions that can be imported by other programs, just like the way you import the **fmt** package that contains functions for printing output to and getting inputs from the console window. In this section, you'll learn how to convert a package into a **module** so that it can be imported into another Go application.

To learn how to create a module, let's create the following directories:

```
$HOME
  |__stringmod
    |__strings
    |__quotes
```

The above creates a module named **stringmod**, with a subdirectory named **strings**. The idea is to group related functionalities into directories so as to logically group them together. This **strings** folder should contain functions related to strings. In this example, **stringmod** is a module and **strings** and **quotes** are packages.

Now, add a file named **strings.go** to the **strings** directory and a file named **quotes.go** to the **quotes** directory:

```
$HOME
  |__stringmod
    |__strings
      |__strings.go
    |__quotes
      |__quotes.go
```

Populate the **strings.go** file with the following:

```
package strings

func internalFunction() {
    // In Go, a name is exported if it
    // begins with a capital letter
}

// Must begin with a capital letter in
// order to be exported
func CountOddEven(s string) (odds,evens int)
{
    odds, evens = 0, 0
    for _, c := range s {
        if int(c) % 2 == 0 {
            evens++
        } else {
            odds++
        }
    }
    return
}
```

Unlike languages like C# and Java, Go has a much simpler approach to access modifiers. Instead of specifying whether a member is *private*, *public*, or *protected*, Go simply uses the function name to determine if a function is exported (visible outside the package) or unexported (restricted to use within the same package). A function name that starts with a capital letter is exported (i.e., can be accessed outside the package) and the rest can only be accessed internally within the package.

Populate the **quotes.go** file with the following:

```
package quotes

import (
    "github.com/hackebrot/turtle"
)

func GetEmoji(name string) string {
    emoji, ok := turtle.Emojis[name]
    if !ok {
        return ""
    }
    return emoji.Char
}
```

Observe that the **quotes** package has a dependency on an external package: **github.com/hackebrot/turtle**.

In Terminal, type the following commands:

```
$ cd ~/stringmod
$ go mod init github.com/weimenglee/stringmod
go: creating new go.mod: module github.com/weimenglee/
stringmod
```

The "**go mod init**" command creates a **go.mod** file in the **stringmod** directory:

```
$HOME
  |__stringmod
    |__go.mod
    |__strings
      |__strings.go
    |__quotes
      |__quotes.go
```

The content of **go.mod** is:

```
module github.com/weimenglee/stringmod
```

The role of the **go.mod** file is to define the module's path, so that it can be imported and used by other packages. Next, type the following command in Terminal to build the module:

```
$ go build
go: finding github.com/hackebrot/turtle v0.1.0
go: downloading github.com/hackebrot/turtle
    v0.1.0
```

During the build process, the package (**github.com/hackebrot/turtle**) required by the **quotes** package is downloaded and installed on your local computer in this path: **~/go/pkg/mod/** directory.

```
$HOME
  |__go
    |__pkg
      |__mod
        |__github.com
          |__hackebrot
            |__turtle
              |__ ...
              |__ ...
```

The **go.mod** file now becomes:

```
module github.com/weimenglee/stringmod

require github.com/hackebrot/turtle v0.1.0
```

It lists all the packages required by the packages inside the module. There's one additional file created: **go.sum**. This file contains the expected cryptographic checksums of the content of specific module versions. It looks like this:

```
github.com/hackebrot/turtle v0.1.0 h1:cmS72nZuooIARtgix6IRPvm
w8r4u8olEZW02Q3DB8YQ=
github.com/hackebrot/turtle v0.1.0/go.mod
h1:vDjX4rgnTSlvROhwGbE2GiB43F/l/8V5TXoRJL2cYTs=
```

Using the Module

With the module created, let's try to import it into another package and use it. Add a new file named **main.go** in the **stringmod** folder:

```
$HOME
  |__stringmod
    |__strings
      |__strings.go
    |__quotes
      |__quotes.go
    |__main.go
```

Populate the **main.go** file as follows:

```
package main

import (
  "fmt"
  "github.com/weimenglee/stringmod/strings"
  "github.com/weimenglee/stringmod/quotes"
)

func main() {
    o, e := strings.CountOddEven("12345")
    fmt.Println(o,e) // 3 2

    fmt.Println(quotes.GetEmoji("turtle"))
}
```

Notice that you're importing the two packages inside the **stringmod** modules using the "**github.com/weimenglee/stringmod**" import path:

```
    "github.com/weimenglee/stringmod/strings"
    "github.com/weimenglee/stringmod/quotes"
```

Also observe that the packages are referred to using their last name in the package path "github.com/weimenglee/stringmod/**strings**" and "github.com/weimenglee/stringmod/**quotes**". If you don't want to use the last name in the package path, you can also provide aliases for the packages during import:

```
package main

import (
  "fmt"
  str
    "github.com/weimenglee/stringmod/strings"
  qt "github.com/weimenglee/stringmod/quotes"
)

func main() {
    o, e := str.CountOddEven("12345")
    fmt.Println(o,e) // 3 2

    fmt.Println(qt.GetEmoji("turtle"))
}
```

Finally, to run the program type the following command in Terminal:

```
$ cd ~/stringmod
$ go run main.go
3 2
🐢
```

Publishing the Module

So far, your module has been created and tested correctly to run locally on your computer. To share it with the world, you simply need to publish it to an online repository, like GitHub. To demonstrate that, I've published the module to GitHub, accessible through the following link: https://github.com/weimenglee/stringmod.

To install this module on your computer, use the following command:

```
$ cd ~
$ go get github.com/weimenglee/stringmod
```

The package is downloaded in the **~/go/src/** and **~/go/bin/** folders:

```
$HOME
  |__go
    |__src
    |  |__github.com
    |     |__hackebrot
    |     |  |__turtle
    |     |     |__ ...
    |     |     |__ ...
    |     |__weimenglee
    |        |__stringmod
    |          |__ ...
    |          |__ ...
    |__bin
       |__stringmod
```

To use the module in your own package, you can import it to your application just like you did in the previous section:

```
package main

import (
    "fmt"
    "github.com/weimenglee/stringmod/quotes"
)

func main() {
    fmt.Println(quotes.GetEmoji("turtle"))
}
```

Go Workspace Directory

In the previous section, you saw that Go uses a number of directories to store your modules and packages. These directories in your **~/go** directory are:

- **Src**: contains the source code of packages that you have installed in your computer
- **Bin**: contains the binary executables of Go applications that have the **main** package (and therefore contains the **main()** function).
- **Pkg**: contains the non-executable packages. These packages are typically imported by other applications.

Summary

By now, you should have a pretty good feel for the Go language. Syntax wise, it's close to C and should be very easy for developers to pick up. Goroutines is one big feature of the language, which should make it a breeze to create multi-threaded server-side apps. Hopefully, this article makes your learning journey much easier and fun!

Wei-Meng Lee

CODE
CONSULTING

TAKE AN HOUR ON US!

Does your team lack the technical knowledge or the resources to start new software development projects, or keep existing projects moving forward? CODE Consulting has top-tier developers available to fill in the technical skills and manpower gaps to make your projects successful. With in-depth experience in .NET, .NET Core, web development, Azure, custom apps for iOS and Android and more, CODE Consulting can get your software project back on track.

Contact us today for a free 1-hour consultation to see how we can help you succeed.

codemag.com/OneHourConsulting
832-717-4445 ext. 9 • info@codemag.com

Using .NET Core Tools to Create Reusable and Shareable Tools and Apps

Starting with .NET Core 2.1, Microsoft introduced the .NET Core Tools platform as part of the .NET Core SDK and since then, these tools have become a vital, although underutilized part of the .NET ecosystem. .NET Core Tools are a simple way to create, publish, and consume what are essentially .NET Core applications that can be published and shared using the existing NuGet infrastructure for packaging and distribution. This means it's quick and easy to build tools that you can share either publicly with the world or privately with yourself or your onsite team.

Rick Strahl
www.west-wind.com
rstrahl@west-wind.com

Rick Strahl is president of West Wind Technologies in Maui, Hawaii. The company specializes in Web and distributed application development and tools, with focus on Windows Server Products, .NET, Visual Studio, and Visual FoxPro. Rick is the author of West Wind Web Connection, West Wind Web Store, and West Wind HTML Help Builder. He's also a C# MVP, a frequent contributor to magazines and books, a frequent speaker at international developer conferences, and the co-publisher of CODE Magazine.

What's a .NET Core Tool?

When you break down a .NET Core Tool into its simplest terms, you end up with this simple statement:

"A .NET Core Tool is a glorified .NET Core application that can be quickly and easily shared and installed via NuGet."

The idea behind a .NET Core Tool is to make it easy to build, publish, and consume executable tools in the same way you can create NuGet packages for .NET and .NET Core components.

Although the original idea was to build tools to aid as part of the build and development process, this platform really offers a much wider scope because **you can publish and share any .NET Core executable application**. This includes servers that run full ASP.NET Core applications or services and even .NET Core desktop applications.

Why Use Dotnet Tools?

Although the idea behind .NET Core Tools isn't anything new, this tooling does provide several benefits to developers and the entire .NET Core ecosystem. The big selling points are:

- Easy to use
  - Single command installs. Example: **dotnet tool install -g LiveReloadServer**
  - Global path access to run command: **LiveReloadServer --help**
- Easy to build
  - Uses standard .NET Core projects
  - Uses existing NuGet infrastructure
  - Nothing new to learn—works with existing tech
  - Configured via standard *.csproj* settings
- Sharing
  - Can be easily shared
  - Can reach a large number of users via NuGet
  - Quickly published and available
  - No explicit package validation
- Community
  - Ease of use and shareability promotes creation of tools
  - Shared content helps build community

On the flip side, there's a big prerequisite to using a .NET Core Tool:

- The **.NET Core SDK is required** to install a .NET Core Tool

The SDK dependence is both a blessing and a curse: Because the .NET Core Runtime is guaranteed to be installed, binaries of your .NET Core Tool can be very small and only consist of application-specific files. But the .NET SDK must exist on the target computer and this SDK install is neither small, nor something that a typical, non-development user already has installed.

Global and Local .NET Core Tools

.NET Core Tools can be installed either as a **global** or **local** tool. Global tools are installed in a central location on the local computer and mapped on the global path, so they are globally accessible. Local tools are installed into a specific folder and only accessible from there. They're essentially project-specific and they-re useful for build systems that need to encapsulate tools as part of a build process or Continuous Integration (CI).

In this article, I focus on the usage for **global tools** and the sharing aspects of .NET Core Tools as general-purpose utilities using the global **-g** command line switch. Everything except the install location and global path access also applies to local tools in the examples.

Careful: Security of .NET Core Tools

Because .NET Cire Tools are executables installed from a remote source and because there's no validation process for published tools, it's important to understand that **there is a potential security risk** to your computer. The code that comes down can execute locally on your system and has access to your local resources when you run the tool.

Be sure you:

- Trust the publisher of the tool.
- Verify that the tool has reviewable source code available in a repository.
- Check for issues on the repository.

Be careful and know the risks!

To be fair, the same cautions apply to NuGet packages because those too can execute any code contained in the package or in a constructor and there's not much concern around that.

.NET Core Tools install and run on your local computer, but they're not validated in the NuGet Package Store, so there's a potential security risk.

There's Nothing New Under the Sun!

"Did you just describe NPM?"

Yup. .NET Core Tools are very much like NPM for .NET Core.

If all this looks familiar from Node,js and NPM—you're right. The idea of shared tools isn't new by any means and follows various other development platforms and their package managers. But for .NET to easily publish and share binary executable tools in a cross-platform manner is relatively new and opens up the reach of small and useful tools that otherwise would never be shared.

What Can You Use .NET Core Tools For?

Although .NET Core Tools were initially designed to provide development time helper tools, they're just plain .NET Core executables that can do anything that a .NET Core executable can do. This means you can use it for all sorts of things that might not be directly developer related.

Here are a few general use cases addressed by .NET Core Tools:

- **Build and Development Tools:** There are many tools that follow the original design goal for creating project helpers that make development tasks easier or facilitate external but related development operations. For example, tools like EF migration commands in **dotnet ef**, **dotnet watch run**, the user secrets manager and **dotnet watch** are all .NET Core Tools that fit this bill. There are many tools available in this category.
- **Generic Command Line Tools:** If you need to build some complex command line helpers that work on scenarios that are more complex than what you reasonably want to do in PowerShell or Bash, a tool can fit that niche nicely. Because these tools can be shared and installed easily and are generally very small, they make a good fit for **beyond scripting** scenarios.
- **Local Servers:** .NET Core makes it easy to build server applications and it's easy to create self-contained Web Server or Services applications. Whether it's running a Web application locally for testing, or whether you have some internal application that maybe is a hybrid using both a Web interface and a desktop application in mixed mode, a .NET Core Tool makes it easy to provide this. It's very powerful to be able to create **small** and easily shareable, self-contained Web and server applications. I'll show a couple of examples of this later.
- **Desktop Applications:** Although Microsoft's official documentation claims that .NET Core Tool is meant for console applications, it turns out that you **can** also create .NET Core WinForms and WPF applications and package them as tools too. But they are Windows-only.

Locating Available .NET Core Tools

One of the reasons that .NET Core Tools aren't so widely used that it's not that easy to find them. Until very recently, you weren't even able to search the NuGet Site specifically for tools but that was recently fixed with new search filters.

There are a couple of other places you can check for tools:

- Nate McMaster has a GitHub repo with a list of many .NET Core Tools at https://tinyurl.com/natetoollist
- The ToolGet Site searches NuGet with a .NET Core Tool filter at https://tinyurl.com/toolget

.NET Core Tool or Platform-Specific Binary?

.NET Core supports creating executable binaries for every platform that it supports. But each platform requires a custom executable launcher and runtime dependencies to run. It takes quite a bit of extra effort to create these separate build targets and distribute each one.

It's possible to create standalone executables for each platform using either a pre-installed .NET Core runtime installation or a fully self-contained executable that can contain all the required runtime files in addition to the files that your application needs to run. Self-contained applications are great for self-reliance and predictable behavior, but they're terribly large, as the .NET Core base runtimes make up a minimum of 70MB of distribution size.

A fully self-contained application is useful and sometimes required. If you're building an end user tool or application, a self-contained application that has everything it needs included is usually a better call. But if you're building developer tools, the lightweight and universal .NET Core Tool distribution experience is often preferable.

The advantage of building a .NET Core Tool is that it's **not platform-specific**. Like a NuGet component, a .NET Core Tool is distributed as a NuGet package that contains only the compiled .NET assemblies. There's no **launching** executable packaged. Because .NET Core Tools rely on an SDK installation to run, the runtime is guaranteed to be there. If **dotnet tool** can run on the computer, so can your .NET Core Tool.

This means that you can build a single, relatively small NuGet package as a .NET Core Tool and it will work on all supported .NET Core platforms, assuming your code is written to otherwise use platform-agnostic features. That's cool: The .NET Core Tool provides you with cross-platform functionality without having to build and maintain multiple platform-specific loaders and it keeps the deployment size very small.

Creating and Using .NET Core Tools

Enough abstract talk! Let's jump in and see how you can:

- Build a .NET Core Tool package for distribution
- Publish a .NET Core Tool package
- Consume a .NET Core Tool package

Create a .NET Core Tool Step-by-Step

Let's create a very simple project called **MagicWindBall** that predicts wind conditions for the day. It's a play on the words of Magic Eightball, which is a fake fortune telling device and I'm hijacking the idea to tell me my wind fortunes.

Start by creating a .NET Core Console project. I'll use the **dotnet** command line tooling, but you can also use Visual Studio (or Rider, etc.) to create a new **.NET Core Console project**.

From the command line, start by creating a new folder with the Project name, and then create a project in that folder:

```
mkdir MagicWindBall
cd MagicWindBall
dotnet new console
code .
```

Source Code on GitHub

You can check out the full MagicWindBall project for this article as well as slides and additional content on the GitHub site. Go to: https://tinyurl.com/dotnettoolsCode

This creates a new project **MagicWindBall.csproj** that you can open in Visual Studio or Visual Studio Code. I'll use VS Code here, as shown in **Figure 1**, which makes it easy to open the projects from a folder via the **code .** command.

This project is very simple, with an imaginary wind forecast generator that randomly displays a string from a list of predefined wind condition strings.

The code is along the lines of **Listing 1** (full code on GitHub at https://tinyurl.com/MagicWindBall).

First, let's make sure the code works as a regular Console application. Open a Terminal in the project folder and do:

```
dotnet run
```

In **Figure 1,** I use the built-in Terminal in VS Code to test the application.

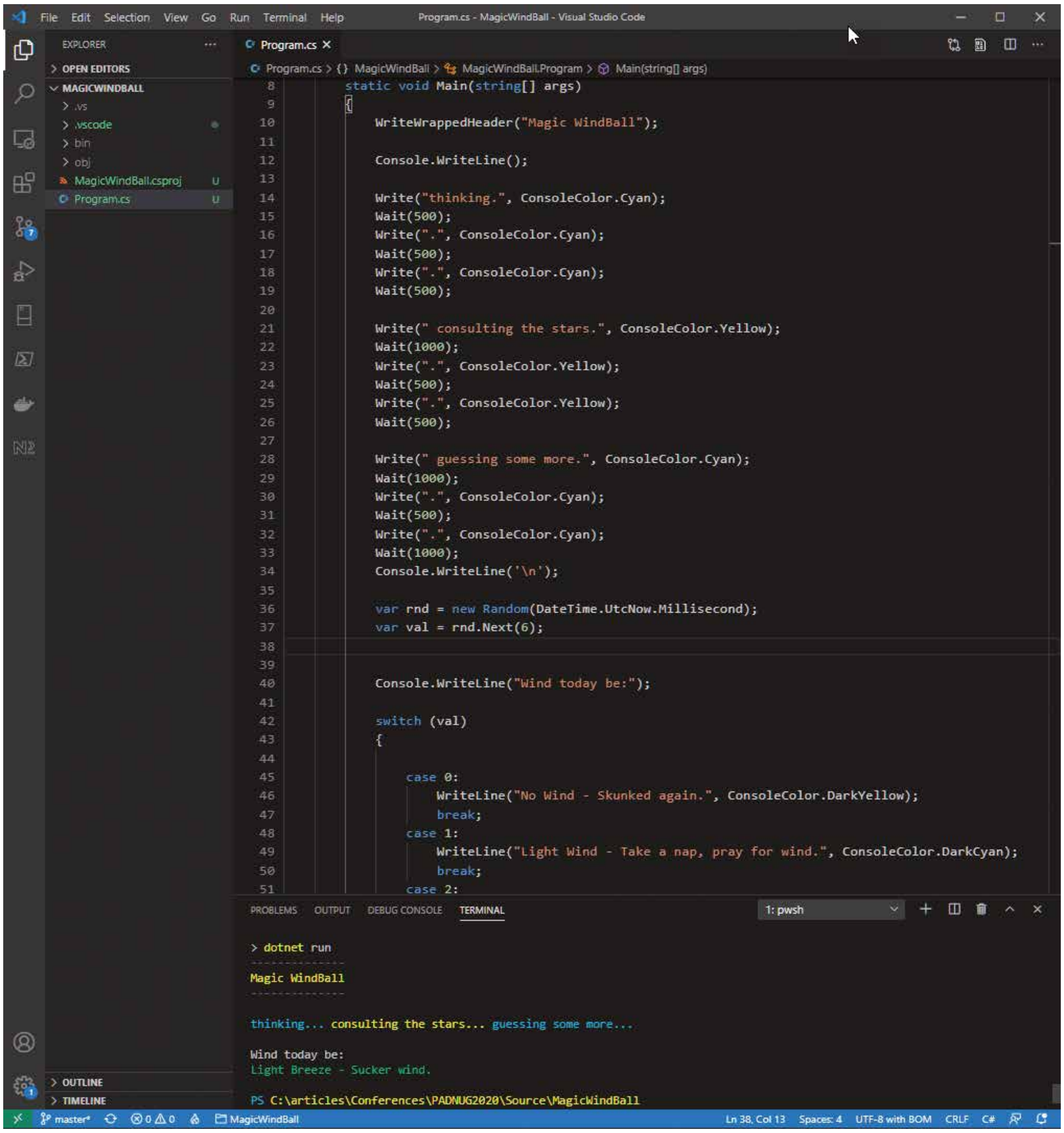
.NET Core Tools use the existing .NET tool infrastructure for creation, publishing, and installing tools.

Make It a .NET Core Tool

You now have a plain .NET Core Console application. To turn this into a .NET Core Tool, add a few specific settings to the project file. At minimum, you need to add a few .NET Core Tool-specific settings to the project. Add **<PackAsTool>**, provide a name, and set up the project to build as a NuGet package, as shown **in Listing 2**.

With these flags in place, you can now build the project and generate the .NET Core Tool NuGet Package into the **./nupkg** folder:

```
dotnet build -c Release
```



```
        static void Main(string[] args)
        {
            WriteWrappedHeader("Magic WindBall");

            Console.WriteLine();

            Write("thinking.", ConsoleColor.Cyan);
            Wait(500);
            Write(".", ConsoleColor.Cyan);
            Wait(500);
            Write(".", ConsoleColor.Cyan);
            Wait(500);

            Write(" consulting the stars.", ConsoleColor.Yellow);
            Wait(1000);
            Write(".", ConsoleColor.Yellow);
            Wait(500);
            Write(".", ConsoleColor.Yellow);
            Wait(500);

            Write(" guessing some more.", ConsoleColor.Cyan);
            Wait(1000);
            Write(".", ConsoleColor.Cyan);
            Wait(500);
            Write(".", ConsoleColor.Cyan);
            Wait(1000);
            Console.WriteLine('\n');

            var rnd = new Random(DateTime.UtcNow.Millisecond);
            var val = rnd.Next(6);


            Console.WriteLine("Wind today be:");

            switch (val)
            {

                case 0:
                    WriteLine("No Wind - Skunked again.", ConsoleColor.DarkYellow);
                    break;
                case 1:
                    WriteLine("Light Wind - Take a nap, pray for wind.", ConsoleColor.DarkCyan);
                    break;
                case 2:
```

```
> dotnet run
--------------
Magic WindBall
--------------

thinking... consulting the stars... guessing some more...

Wind today be:
Light Breeze - Sucker wind.

PS C:\articles\Conferences\PADNUG2020\Source\MagicWindBall
```

Figure 1: Opening and running the Console project in Visual Studio Code.

Listing 1: The MagicWindBall Sample Tool

```
static void Main(string[] args)
{
    WriteWrappedHeader("Magic WindBall");

    Console.WriteLine();

    Write("thinking.", ConsoleColor.Cyan);
    Wait(500);

    Write(" consulting the stars.", ConsoleColor.Yellow);
    Wait(1000);
    Write(".", ConsoleColor.Yellow);

    Write(" guessing some more.", ConsoleColor.Cyan);
    Wait(1000);
    Write(".", ConsoleColor.Cyan);

    Console.WriteLine('\n');

    var rnd = new Random(DateTime.UtcNow.Millisecond);
    var val = rnd.Next(6);

    Console.WriteLine("Wind today be:");

    switch (val)
    {

        case 0:
            WriteLine("No Wind - Skunked again.",
                      ConsoleColor.DarkYellow);
            break;
        case 1:
            WriteLine("Light Wind - Pray for wind.",
                      ConsoleColor.DarkCyan);
            break;
        case 2:
            WriteLine("Light Breeze - Sucker wind.",
                      ConsoleColor.DarkGreen);
            break;
        case 3:
            WriteLine("Breezy- Wake up and get ready.",
                      ConsoleColor.Green);
            break;
        case 4:
            WriteLine("Windy- Why are you still here?",
                      ConsoleColor.Yellow);
            break;
        case 5:
            WriteLine("Super Nuker- Batten down the hatches",
                      ConsoleColor.Red);
            break;
        default:
            WriteLine("Roll the Dice- Coming in waves.");
            break;
    }

    Console.WriteLine();
}
```

Listing 2: Adding Dotnet Tool specific .csproj settings

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp3.1</TargetFramework>
  </PropertyGroup>

  <!-- Dotnet Tool Specific settings -->
  <PropertyGroup>
    <PackAsTool>true</PackAsTool>
    <PackageId>dotnet-magicwindball</PackageId>
    <ToolCommandName>magicwindball</ToolCommandName>

    <PackageOutputPath>./nupkg</PackageOutputPath>
    <GeneratePackageOnBuild>true</GeneratePackageOnBuild>
  </PropertyGroup>

</Project>
```

If you use Visual Studio, just build in **Release** mode.

This builds the project and creates a NuGet Package in the **./nupkg** folder. You can use the **NuGet Package Explorer** to spy into the package to see what's in the package, as shown in **Figure 2**.

Testing the .NET Core Tool Locally

Once you've created the NuGet Package, you'll probably want to test it locally first before publishing it. You can do that by installing the .NET Core Tool locally from a folder. To install, you use **dotnet tool install** or **dotnet tool update**.

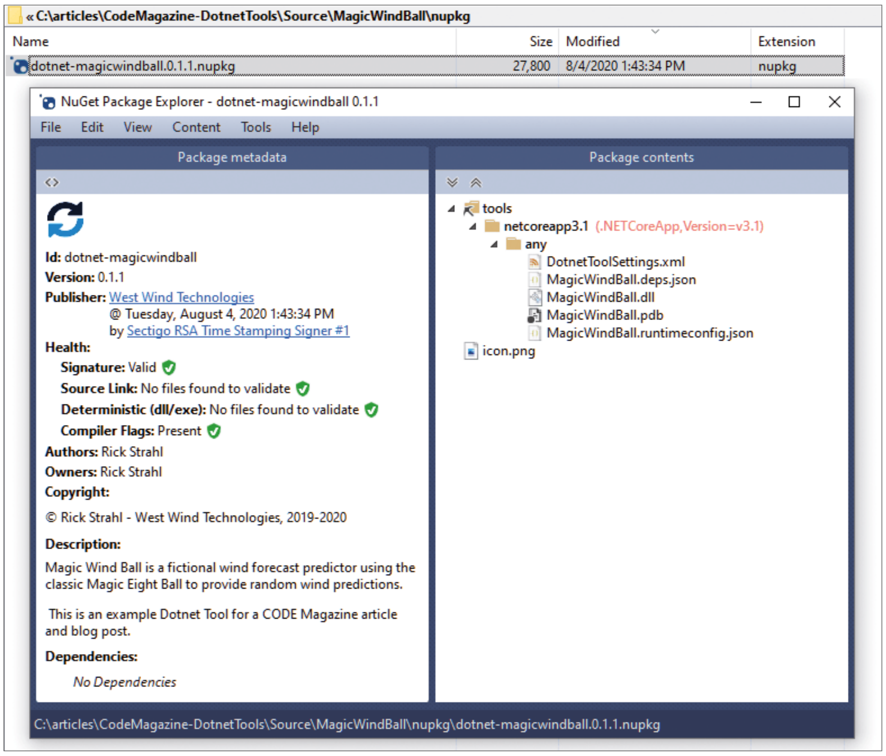
For public tools that come from the default cloud based NuGet package store, you use:

```
dotnet tool install =g dotnet-magicwindball
```

But…it doesn't work yet because I haven't published the package yet. You can test the package locally by installing the NuGet component from a folder by specifying the **--add-source ./nupkg** command line option:

```
dotnet tool install -g dotnet-magicwindball
                    add-source ./nupkg
```

Et voila! You've just installed the .NET Core Tool locally and you can now run the tool simply by typing **magicwindball**


« C:\articles\CodeMagazine-DotnetTools\Source\MagicWindBall\nupkg

Name	Size	Modified	Extension
dotnet-magicwindball.0.1.1.nupkg	27,800	8/4/2020 1:43:34 PM	nupkg

NuGet Package Explorer - dotnet-magicwindball 0.1.1

Id: dotnet-magicwindball
Version: 0.1.1
Publisher: West Wind Technologies
@ Tuesday, August 4, 2020 1:43:34 PM
by Sectigo RSA Time Stamping Signer #1
Health:
Signature: Valid
Source Link: No files found to validate
Deterministic (dll/exe): No files found to validate
Compiler Flags: Present
Authors: Rick Strahl
Owners: Rick Strahl
Copyright:
© Rick Strahl - West Wind Technologies, 2019-2020
Description:
Magic Wind Ball is a fictional wind forecast predictor using the classic Magic Eight Ball to provide random wind predictions.

This is an example Dotnet Tool for a CODE Magazine article and blog post.
Dependencies:
No Dependencies

```
tools
  netcoreapp3.1 (.NETCoreApp,Version=v3.1)
    any
      DotnetToolSettings.xml
      MagicWindBall.deps.json
      MagicWindBall.dll
      MagicWindBall.pdb
      MagicWindBall.runtimeconfig.json
icon.png
```

C:\articles\CodeMagazine-DotnetTools\Source\MagicWindBall\nupkg\dotnet-magicwindball.0.1.1.nupkg

Figure 2: The .NET Core Tool NuGet Package in the NuGet Package Explorer.

into the Terminal from anywhere on your computer. **Figure 3** shows building, installing, and running the tool in a Terminal window.

Publishing to NuGet

.NET Core Tools are published as NuGet packages and use the same exact mechanism you use to publish a NuGet package for a .NET Component. You use **NuGet publish** or you can also use the NuGet Package Explorer shown earlier in **Figure 2**.

Here's a build, sign, and publish PowerShell script (the full script is here: https://tinyurl.com/SnippetConverter):

```
dotnet build -c Release

$filename = gci "./nupkg/*.nupkg" | `
            sort LastWriteTime | `
            select -last 1 | `
            select -ExpandProperty "Name"
$len = $filename.length

if ($len -gt 0) {
   nuget sign  ".\nupkg\$filename"   `
        -CertificateSubject "West Wind " `
        -timestamper "..."
   nuget push  ".\nupkg\$filename"
        -source nuget.org
}
```

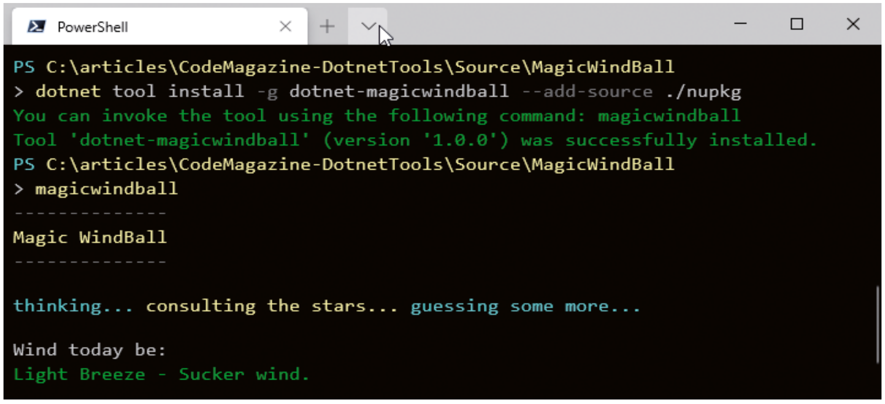
Note that NuGet Package signing is optional, but because I already have a publisher certificate, I'm using it to sign my package. For the **nuget push** to work, you'll need to set the active NuGet publishing ID before you publish the package.

```
nuget setApiKey <your_API_key>
```

Figure 4 shows what the published package looks like on the NuGet Package Store site.

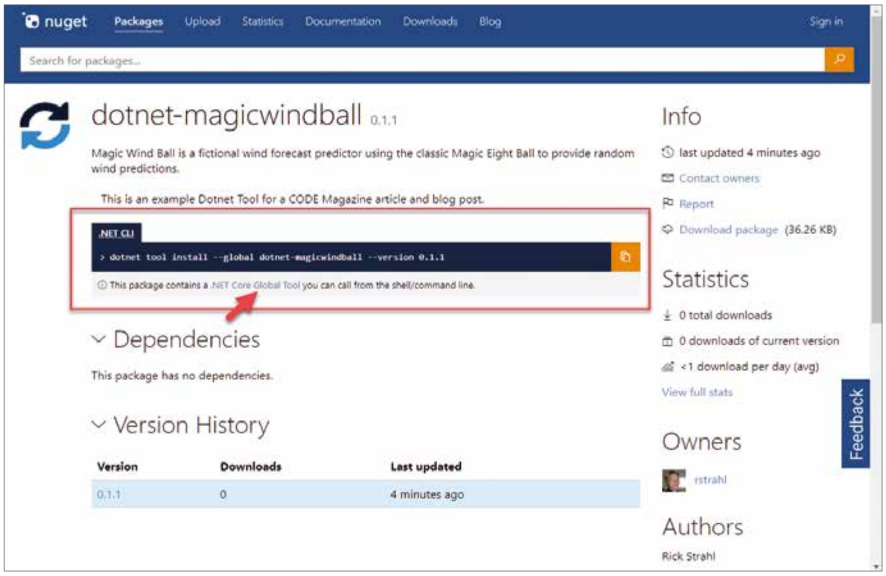
Once published, the package becomes accessible within a few minutes. Feed listings can take a bit longer, so you may have to explicitly specify a version on the command line using the **--version** flag.

You can now install and run the component from the NuGet Package Store. **Figure 5** shows the install and run sequence.



```
PS C:\articles\CodeMagazine-DotnetTools\Source\MagicWindBall
> dotnet tool install -g dotnet-magicwindball --add-source ./nupkg
You can invoke the tool using the following command: magicwindball
Tool 'dotnet-magicwindball' (version '1.0.0') was successfully installed.
PS C:\articles\CodeMagazine-DotnetTools\Source\MagicWindBall
> magicwindball
-------------
Magic WindBall
-------------

thinking... consulting the stars... guessing some more...

Wind today be:
Light Breeze - Sucker wind.
```

Figure 3: Build and run your .NET Core Tool locally.



dotnet-magicwindball 0.1.1

Magic Wind Ball is a fictional wind forecast predictor using the classic Magic Eight Ball to provide random wind predictions.

This is an example Dotnet Tool for a CODE Magazine article and blog post.

```
> dotnet tool install --global dotnet-magicwindball --version 0.1.1
```

This package contains a .NET Core Global Tool you can call from the shell/command line.

Dependencies

This package has no dependencies.

Version History

Version	Downloads	Last updated
0.1.1	0	4 minutes ago

Info

last updated 4 minutes ago

Contact owners

Report

Download package (36.26 KB)

Statistics

0 total downloads

0 downloads of current version

<1 download per day (avg)

View full stats

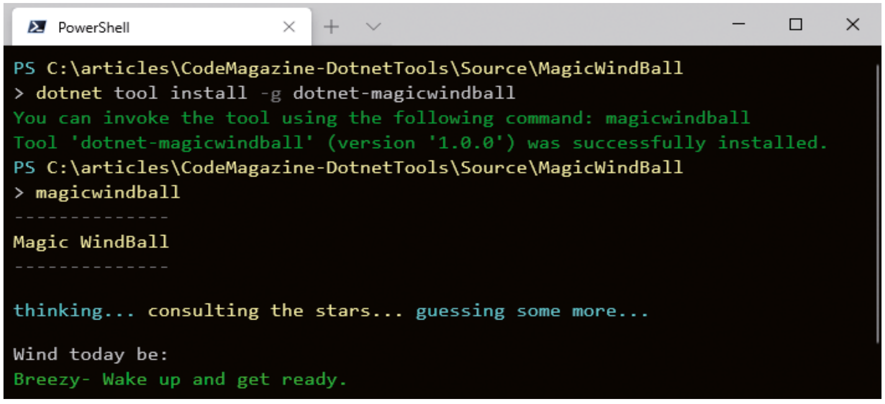
Owners

rstrahl

Authors

Rick Strahl

Figure 4: The published .NET Core Tool in the NuGet Package Store.



```
PS C:\articles\CodeMagazine-DotnetTools\Source\MagicWindBall
> dotnet tool install -g dotnet-magicwindball
You can invoke the tool using the following command: magicwindball
Tool 'dotnet-magicwindball' (version '1.0.0') was successfully installed.
PS C:\articles\CodeMagazine-DotnetTools\Source\MagicWindBall
> magicwindball
-------------
Magic WindBall
-------------

thinking... consulting the stars... guessing some more...

Wind today be:
Breezy- Wake up and get ready.
```

Figure 5: Install and run a .NET Core Tool.

> .NET Core Tools create a platform-specific executable launched in a globally accessible folder to make the tool available globally from anywhere on the computer.

If you need to update the .NET Core Tool, make your code changes and increment the version number of the project then simply re-publish to NuGet. A new package with the new version number is created and pushed, which then becomes available on NuGet. You can then use **dotnet tool update -g dotnet-magicwindball** to update the local tool installation to the updated version.

How a Tool Gets Executed

.NET Core Tools are deployed as .NET NuGet packages that **don't include an OS specific executable file**. Rather, when a tool is installed, a platform-specific, native launcher executable is generated that acts as a proxy launcher for the .NET Core runtime, which then bootstraps the .NET Core Tool entry assembly. The launcher is created in a system mapped **.dotnet** folder and is then globally available using the **<Command-Name>** specified by the project. **Figure 6** shows the launcher in the **.dotnet** root folder and the actual install folder that holds only the .NET assemblies that execute on any platform.

The **.dotnet/.store** path holds the actual unpacked NuGet package content for each tool installed. When you run the launcher (**magicwindball.exe** on Windows), it loads the .NET Core runtime and then calls the **static void Main()** entry point in the entry assembly of the package. The **.exe** you see on the left in **Figure 6** is only a stub launcher. If you hook up an IL decompiler to the EXE as I've done in **Figure 6** with Reflector, you'll find that the exe is a **native binary**, not a .NET assembly.

Running on Another Platform: Linux with WSL and Mac

.NET Core Tools are platform agnostic and, assuming your application doesn't use any platform-specific features, they can run on Windows, Mac, or Linux from a single distribution.

So, let's run this .NET Core Tool on Linux using Windows Subsystem for Linux (WSL) using the same steps as before. Use **dotnet tool install** and then execute the command. **Figure 7** shows what that looks like inside of WSL. You can do the same on an actual Linux or Mac computer. **Figure 8** shows running on a Mac.

It all works on all .NET Core supported platforms from the single .NET Core Tool package, as long as your code doesn't do anything platform-specific.

> .NET Core Tools don't need platform-specific builds: They run on any supported .NET Core platform from a single code base and NuGet package.

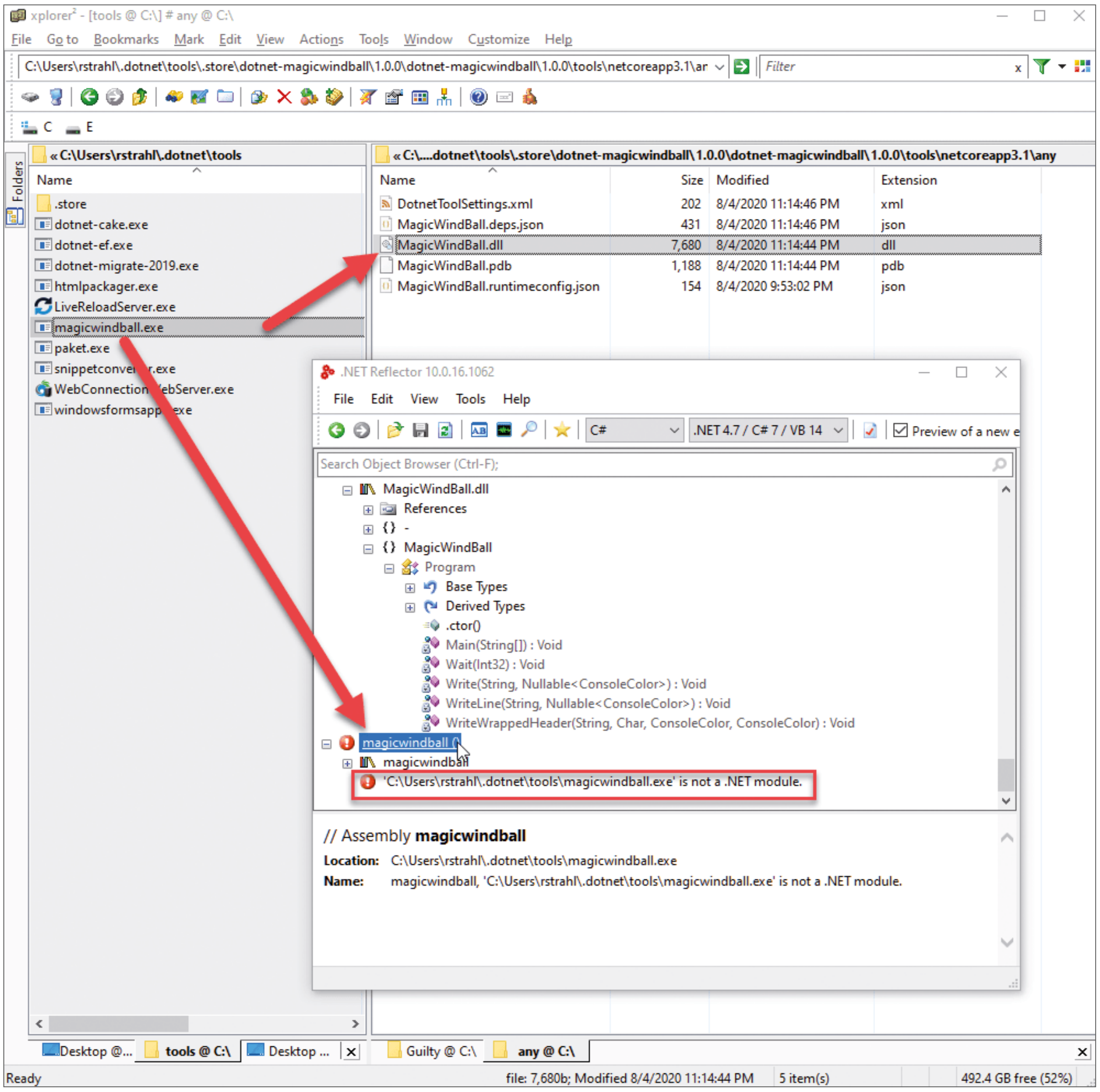
Listing and Managing Installed Tools

You can check what tools you have installed by using the **dotnet tool list -g** command, as shown in **Figure 9**.

Searching NuGet for Tools

Until very recently, the NuGet website didn't support searching specifically for .NET Core Tools. This has been addressed very recently via a new **Filter** feature that you can use on the results page to filter the result list. It's not exactly intuitive to do this after the search completes but at least now you can filter by .NET Core Tool. Go to: https://nuget.org

For a good and frequently updated list of many .NET Core Tools available, go check out Nate McMaster's .NET Core Tool List: https://tinyurl.com/natetoollist


xplorer² - [tools @ C:\] # any @ C:\
File Go to Bookmarks Mark Edit View Actions Tools Window Customize Help
Folders
Desktop @... tools @ C:\ Desktop ... Guilty @ C:\ any @ C:\
.NET Reflector 10.0.16.1062
File Edit View Tools Help
C# .NET 4.7 / C# 7 / VB 14 Preview of a new e
Search Object Browser (Ctrl-F);

C:\Users\rstrahl\.dotnet\tools\.store\dotnet-magicwindball\1.0.0\dotnet-magicwindball\1.0.0\tools\netcoreapp3.1\ar

« C:\Users\rstrahl\.dotnet\tools

Name
.store
dotnet-cake.exe
dotnet-ef.exe
dotnet-migrate-2019.exe
htmlpackager.exe
LiveReloadServer.exe
magicwindball.exe
paket.exe
snippetconverter.exe
WebConnectionWebServer.exe
windowsformsapp.exe

« C:\...dotnet\tools\.store\dotnet-magicwindball\1.0.0\dotnet-magicwindball\1.0.0\tools\netcoreapp3.1\any

Name	Size	Modified	Extension
DotnetToolSettings.xml	202	8/4/2020 11:14:46 PM	xml
MagicWindBall.deps.json	431	8/4/2020 11:14:46 PM	json
MagicWindBall.dll	7,680	8/4/2020 11:14:44 PM	dll
MagicWindBall.pdb	1,188	8/4/2020 11:14:44 PM	pdb
MagicWindBall.runtimeconfig.json	154	8/4/2020 9:53:02 PM	json

```
MagicWindBall.dll
  References
  {} -
  {} MagicWindBall
    Program
      Base Types
      Derived Types
      .ctor()
      Main(String[]) : Void
      Wait(Int32) : Void
      Write(String, Nullable<ConsoleColor>) : Void
      WriteLine(String, Nullable<ConsoleColor>) : Void
      WriteWrappedHeader(String, Char, ConsoleColor, ConsoleColor) : Void
magicwindball (
  magicwindball
  'C:\Users\rstrahl\.dotnet\tools\magicwindball.exe' is not a .NET module.
```

```
// Assembly magicwindball
Location: C:\Users\rstrahl\.dotnet\tools\magicwindball.exe
Name:     magicwindball, 'C:\Users\rstrahl\.dotnet\tools\magicwindball.exe' is not a .NET module.
```

Ready   file: 7,680b; Modified 8/4/2020 11:14:44 PM   5 item(s)   492.4 GB free (52%)

Figure 6: An installed .NET Core Tool uses a proxy executable to launch the .NET Core Tool.

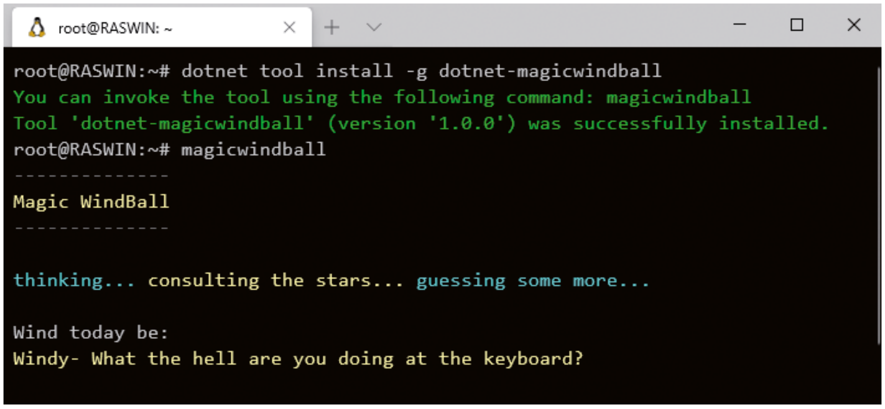

```
root@RASWIN:~# dotnet tool install -g dotnet-magicwindball
You can invoke the tool using the following command: magicwindball
Tool 'dotnet-magicwindball' (version '1.0.0') was successfully installed.
root@RASWIN:~# magicwindball
--------------
Magic WindBall
--------------

thinking... consulting the stars... guessing some more...

Wind today be:
Windy- What the hell are you doing at the keyboard?
```

Figure 7: .NET Core Tools are cross-platform, here running under Linux in WSL.

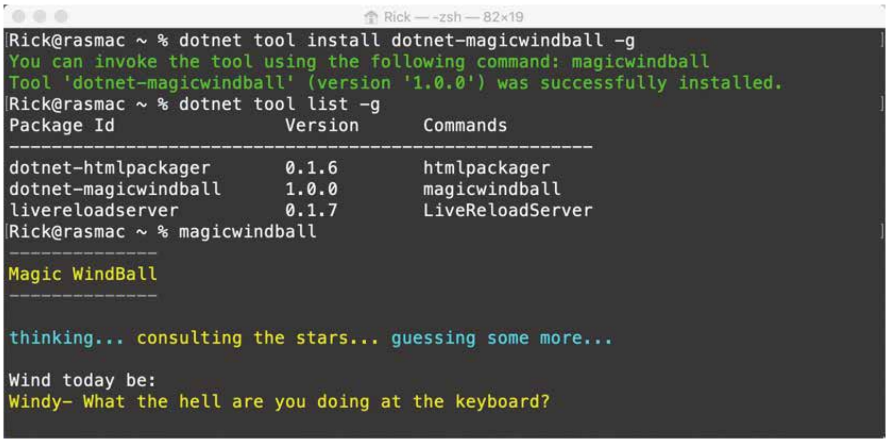

```
Rick@rasmac ~ % dotnet tool install dotnet-magicwindball -g
You can invoke the tool using the following command: magicwindball
Tool 'dotnet-magicwindball' (version '1.0.0') was successfully installed.
Rick@rasmac ~ % dotnet tool list -g
Package Id                Version      Commands
-------------------------------------------------------
dotnet-htmlpackager       0.1.6        htmlpackager
dotnet-magicwindball      1.0.0        magicwindball
livereloadserver          0.1.7        LiveReloadServer
Rick@rasmac ~ % magicwindball
--------------
Magic WindBall
--------------

thinking... consulting the stars... guessing some more...

Wind today be:
Windy- What the hell are you doing at the keyboard?
```

Figure 8: .NET Core Tools are cross-platform, here running under Mac in WSL.

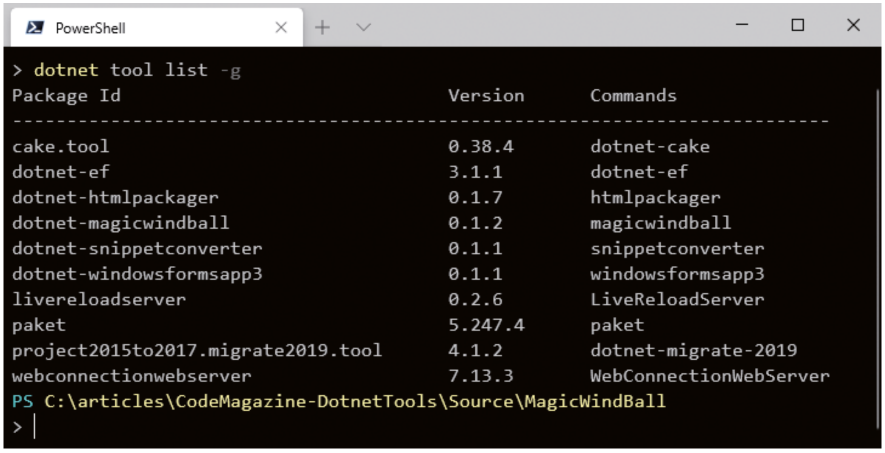

```
> dotnet tool list -g
Package Id                             Version      Commands
--------------------------------------------------------------------------
cake.tool                              0.38.4       dotnet-cake
dotnet-ef                              3.1.1        dotnet-ef
dotnet-htmlpackager                    0.1.7        htmlpackager
dotnet-magicwindball                   0.1.2        magicwindball
dotnet-snippetconverter                0.1.1        snippetconverter
dotnet-windowsformsapp3                0.1.1        windowsformsapp3
livereloadserver                       0.2.6        LiveReloadServer
paket                                  5.247.4      paket
project2015to2017.migrate2019.tool     4.1.2        dotnet-migrate-2019
webconnectionwebserver                 7.13.3       WebConnectionWebServer
PS C:\articles\CodeMagazine-DotnetTools\Source\MagicWindBall
> |
```

Figure 9: Listing installed .NET Core Tool components.

The list shows both the **Package Id,** which is the install name used with NuGet, and the **Command** name, which is used to run the package. Package ID and Command can be different, as demonstrated by the **dotnet-magicwindball** ID and the **magicwindball** command. But they don't have to be different. For example, my **LiveReloadServer** package uses the same name for the package ID and command name. The **dotnet-** prefix is an original convention used by various .NET internal tools to make it obvious that they're .NET tools, but that convention is discouraged now. Generally, it's better to use the same name for the package ID and command and to use the most descriptive name possible.

Keep in mind that tool command names are **case sensitive** on case sensitive operating systems.

You can keep tools up to date using the **update** command:

```
dotnet tool update -g dotnet-magicwindball
```

The **update** command also installs a tool if it isn't already installed. You can also easily uninstall tools with **uninstall**:

```
dotnet tool uninstall -g `
        dotnet-magicwindball
```

Example Components

To bring home the utility of .NET Core Tools, I'd like to describe a few of the .NET Core Tools I've created and discuss the hows and whys of creating them. I've been huge fan of these tools because it's so frictionless to publish a tool for easy reuse. If it weren't for the ease of sharing via NuGet, I probably wouldn't have bothered sharing these tools at all.

Some of these tools I built mainly for myself, but because I've made them public, they ended up getting used by quite a few other people. Maybe you'll find some of these useful as well. But more importantly, I hope it inspires you to share your own tools no matter how silly or simple—somebody else might find them useful too!

Simple Tools

The first couple of examples are your typical utility tools that, under normal circumstances, I would have distributed as a downloadable installer or just a standalone binary. If a tool is popular enough, I still go that route **in addition to the .NET Core Tool**, but I tend to use the .NET Core Tool rather than a standalone installed application.

Visual Studio Snippet Converter

This tool (https://tinyurl.com/SnippetConverter) is a small utility to convert Visual Studio Code Snippets (code expansions) to other development environments. It supports:

- Visual Studio Code Snippets
- JetBrains Rider Snippets

This tool is directly targeted at developers who are already using Visual Studio and so are **very likely** to have the .NET SDK installed just by virtue of using Visual Studio. As a tool, this is perfect, and it fits the **.NET Core Project Tooling** use case that was the original design goal by Microsoft.

The background behind this tool is that I have a ton of Visual Studio code snippets that help me quickly format blocks of code—from inserting properties, creating entire blocks of classes that have complex signatures or inserting complex HTML markup completions for various Web frameworks. Snippets are an incredibly useful and very underutilized feature in various IDEs.

I use several different development tools: Visual Studio, VS Code, and Rider, and sharing snippets across them is much easier than recreating them in each environment.

This tool lets me export my Visual Studio snippets into VS Code and Rider. It also provides me use a "master" snippet repository in Visual Studio that allows me to update snippets there and then easily update the snippets in the other environments with the .NET Core Tool.

To use this tool:

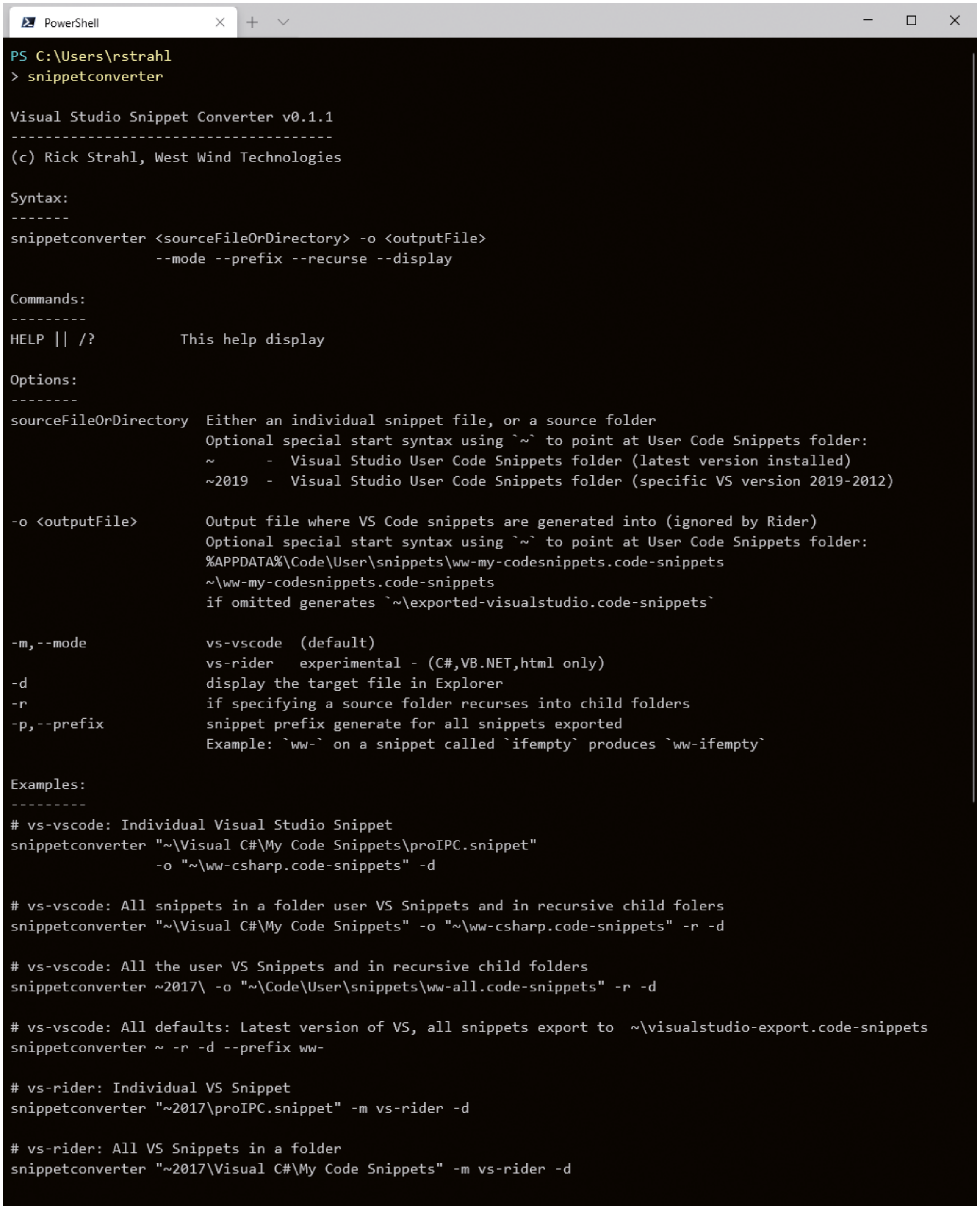
```
dotnet Tool install -g snippetconverter
```

For options, just run it and it shows the options shown in **Figure 10**, which illustrates the functionality best.

To run the converter, you can specify a source snippet or folder and an output path:

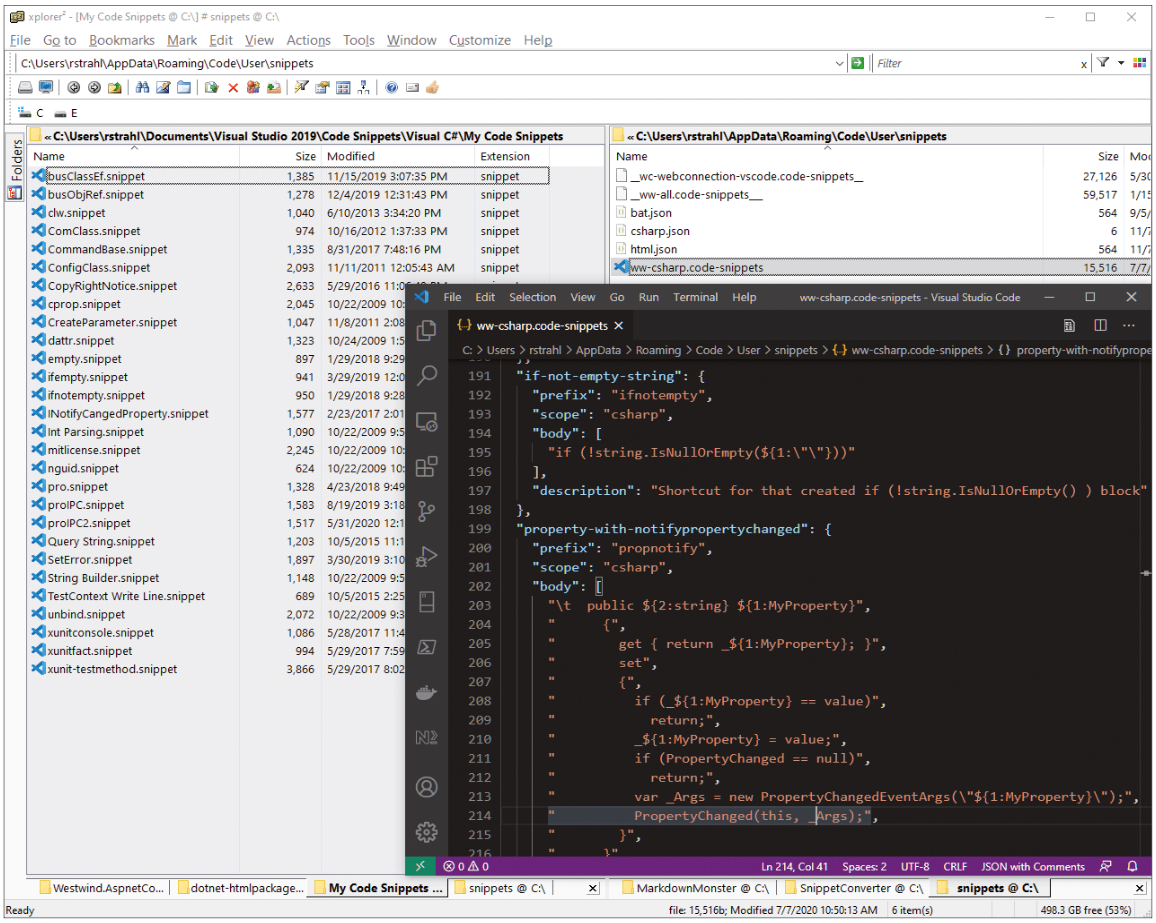
```
snippetconverter `
      "~\Visual C#\My Code Snippets" `
      -o "~\ww-csharp.code-snippets" `
      -m vs-vscode -r -d
```

To make the snippet location easier to use, the tool lets you use ~ for the default snippet folders for each IDE. For Visual Studio, this points to the **<Documents>\Visual Studio 2019 Code Snippets** folder, for example.


PowerShell

```
PS C:\Users\rstrahl
> snippetconverter

Visual Studio Snippet Converter v0.1.1
--------------------------------------
(c) Rick Strahl, West Wind Technologies

Syntax:
-------
snippetconverter <sourceFileOrDirectory> -o <outputFile>
                 --mode --prefix --recurse --display

Commands:
---------
HELP || /?          This help display

Options:
--------
sourceFileOrDirectory  Either an individual snippet file, or a source folder
                       Optional special start syntax using `~` to point at User Code Snippets folder:
                       ~      -  Visual Studio User Code Snippets folder (latest version installed)
                       ~2019  -  Visual Studio User Code Snippets folder (specific VS version 2019-2012)

-o <outputFile>        Output file where VS Code snippets are generated into (ignored by Rider)
                       Optional special start syntax using `~` to point at User Code Snippets folder:
                       %APPDATA%\Code\User\snippets\ww-my-codesnippets.code-snippets
                       ~\ww-my-codesnippets.code-snippets
                       if omitted generates `~\exported-visualstudio.code-snippets`

-m,--mode              vs-vscode  (default)
                       vs-rider   experimental - (C#,VB.NET,html only)
-d                     display the target file in Explorer
-r                     if specifying a source folder recurses into child folders
-p,--prefix            snippet prefix generate for all snippets exported
                       Example: `ww-` on a snippet called `ifempty` produces `ww-ifempty`

Examples:
---------
# vs-vscode: Individual Visual Studio Snippet
snippetconverter "~\Visual C#\My Code Snippets\proIPC.snippet"
                 -o "~\ww-csharp.code-snippets" -d

# vs-vscode: All snippets in a folder user VS Snippets and in recursive child folers
snippetconverter "~\Visual C#\My Code Snippets" -o "~\ww-csharp.code-snippets" -r -d

# vs-vscode: All the user VS Snippets and in recursive child folders
snippetconverter ~2017\ -o "~\Code\User\snippets\ww-all.code-snippets" -r -d

# vs-vscode: All defaults: Latest version of VS, all snippets export to  ~\visualstudio-export.code-snippets
snippetconverter ~ -r -d --prefix ww-

# vs-rider: Individual VS Snippet
snippetconverter "~2017\proIPC.snippet" -m vs-rider -d

# vs-rider: All VS Snippets in a folder
snippetconverter "~2017\Visual C#\My Code Snippets" -m vs-rider -d
```

Figure 10: A Snippet Converter tool to convert Visual Studio Snippets to VS Code and Rider.


xplorer² - [My Code Snippets @ C:\] # snippets @ C:\
File Go to Bookmarks Mark Edit View Actions Tools Window Customize Help
Filter
C E
Folders

C:\Users\rstrahl\AppData\Roaming\Code\User\snippets

«C:\Users\rstrahl\Documents\Visual Studio 2019\Code Snippets\Visual C#\My Code Snippets

Name	Size	Modified	Extension
busClassEf.snippet	1,385	11/15/2019 3:07:35 PM	snippet
busObjRef.snippet	1,278	12/4/2019 12:31:43 PM	snippet
clw.snippet	1,040	6/10/2013 3:34:20 PM	snippet
ComClass.snippet	974	10/16/2012 1:37:33 PM	snippet
CommandBase.snippet	1,335	8/31/2017 7:48:16 PM	snippet
ConfigClass.snippet	2,093	11/11/2011 12:05:43 AM	snippet
CopyRightNotice.snippet	2,633	5/29/2016 11:0	
cprop.snippet	2,045	10/22/2009 10:	
CreateParameter.snippet	1,047	11/8/2011 2:08	
dattr.snippet	1,323	10/24/2009 1:5	
empty.snippet	897	1/29/2018 9:29	
ifempty.snippet	941	3/29/2019 12:0	
ifnotempty.snippet	950	1/29/2018 9:28	
INotifyCangedProperty.snippet	1,577	2/23/2017 2:01	
Int Parsing.snippet	1,090	10/22/2009 9:5	
mitlicense.snippet	2,245	10/22/2009 10:	
nguid.snippet	624	10/22/2009 10:	
pro.snippet	1,328	4/23/2018 9:49	
proIPC.snippet	1,583	8/19/2019 3:18	
proIPC2.snippet	1,517	5/31/2020 12:1	
Query String.snippet	1,203	10/5/2015 11:1	
SetError.snippet	1,897	3/30/2019 3:10	
String Builder.snippet	1,148	10/22/2009 9:5	
TestContext Write Line.snippet	689	10/5/2015 2:25	
unbind.snippet	2,072	10/22/2009 9:3	
xunitconsole.snippet	1,086	5/28/2017 11:4	
xunitfact.snippet	994	5/29/2017 7:59	
xunit-testmethod.snippet	3,866	5/29/2017 8:02	

«C:\Users\rstrahl\AppData\Roaming\Code\User\snippets

Name	Size	Mod
__wc-webconnection-vscode.code-snippets__	27,126	5/30
__ww-all.code-snippets__	59,517	1/15
bat.json	564	9/5/
csharp.json	6	11/7
html.json	564	11/7
ww-csharp.code-snippets	15,516	7/7/

File Edit Selection View Go Run Terminal Help
ww-csharp.code-snippets - Visual Studio Code
ww-csharp.code-snippets
C: > Users > rstrahl > AppData > Roaming > Code > User > snippets > {..} ww-csharp.code-snippets > {} property-with-notifyprope

```
"if-not-empty-string": {
  "prefix": "ifnotempty",
  "scope": "csharp",
  "body": [
    "if (!string.IsNullOrEmpty(${1:\"\"}))"
  ],
  "description": "Shortcut for that created if (!string.IsNullOrEmpty() ) block"
},
"property-with-notifypropertychanged": {
  "prefix": "propnotify",
  "scope": "csharp",
  "body": [
    "\t  public ${2:string} ${1:MyProperty}",
    "       {",
    "         get { return _${1:MyProperty}; }",
    "         set",
    "         {",
    "           if (_${1:MyProperty} == value)",
    "             return;",
    "           _${1:MyProperty} = value;",
    "           if (PropertyChanged == null)",
    "             return;",
    "           var _Args = new PropertyChangedEventArgs(\"${1:MyProperty}\");",
    "           PropertyChanged(this, _Args);",
    "         }",
```

⊗ 0 ⚠ 0
Ln 214, Col 41 Spaces: 2 UTF-8 CRLF JSON with Comments
Westwind.AspnetCo... dotnet-htmlpackage... My Code Snippets ... snippets @ C:\
MarkdownMonster @ C:\ SnippetConverter @ C:\ snippets @ C:\
Ready

file: 15,516b; Modified 7/7/2020 10:50:13 AM 6 item(s) 498.3 GB free (53%)

Figure 11: Migrated code snippets from the Snippet Converter in VS Code.

Figure 11 shows the source snippet folder and output generated by the Visual Studio code export:

HtmlPackager

HtmlPackager (https://tinyurl.com/HtmlPackager) is a tool to package an HTML URL into a fully self-contained *"package"* or folder, bringing all the content offline so it can be viewed completely offline. It's similar to **Save as...** in a Web browser, but you can automate the process. This is useful for attaching to emails or for archival purposes.

This tool came about because I needed this functionality at the time for my Markdown Monster (https://markdownmonster.west-wind.com) editor. I looked around for a command line tool but came up with nothing that worked that had a small footprint and no dependencies. As a result, I ended up building a .NET library that I used in Markdown Monster initially, but then also built into a .NET Core Tool so it can be used from the command line and automated.

It's a general-purpose tool and **.NET Core Tooling** provides a very easy way to publish and share it.

The **dotnet-htmlpackager** tool can be installed with:

```
dotnet tool install -g dotnet-htmlpackager
```

Once installed, you can run it using the **help** command to see the command line options. To capture a URL, it uses a command like this:

```
htmlpackager  https://west-wind.com `
  -v -o /temp/captured/MarkdownMonster.html
```

This creates a single very large but fully self-contained HTML file as shown in **Figure 12**.

There are other options for creating:

- A folder of loose HTML and resource files (**-x**)
- A zip file of the folder of loose HTML and resources (**-z**)

Servers

A .NET Core Tool is just a .NET Core application, so you can take advantage of just about any .NET Core feature, including the ability to create self-contained Web server applications and services that you can run locally.

LiveReloadServer: A Generic Static File Web Server

I frequently run local static websites on my development computer for quickly running a SPA application, working on static website content or one of my many standalone JavaScript library projects.

I could spin up a full development environment like Visual Studio and IIS Express, or use something like WebPack to provide infrastructure for hosting and running, but that but that feels like overkill for many of the simple run and edit scenarios I have. As an alternative, using the **LiveReloadServer** (https://tinyurl.com/LiveReloadServer), I can quickly run a local development server and just run a local folder

.NET Core Tools and Windows Apps

It's possible to publish .NET Core Windows Forms and WPF applications as .NET Core Tools with the .NET Core 3.x SDKs. However, they're limited to Windows as they are obviously Windows-specific technologies.

Additionally, in .NET 5.0, the Windows .NET Core runtimes won't be automatically installed as part of an SDK install as they are today in the .NET Core 3.x SDKs, so there's no guarantee of the required runtimes being available in the future. If that's the case the utility of a .NET Core Tool for these technologies is somewhat diminished.

as a website. It's a small, self-contained .NET Core-based static Web Server that can serve local static file content out of an arbitrary folder that you specify. This server can have:

- Local static Web content
- Loose Razor pages (optional)
- Markdown pages (optional)

In addition, it also provides what, for me, is the most useful feature:

- Live reload of changed content (optional)

The live reload functionality is built-in, enabled by default, and automatically refreshes any active HTML pages loaded through the server when an HTML page or any other related resource like CSS or JavaScript are changed. This makes it a great, generic tool to use on older non-build process websites or tools you might need to run locally to add functionality and make changes.

As a .NET Core Tool, it's easy to install the server and it's very quick to run and start up.

A standalone local Web server isn't a new idea, as there have been static Node.js-based servers like http-server (https://github.com/http-party/http-server) or browser-sync (https://www.browsersync.io/) forever, but these tools require Node,js and the standalone live reload tools like **BrowserSync** I've used don't work very reliably. LiveReloadServer is an alternative and it addresses all these scenarios nicely.

To use Live Reload Server:

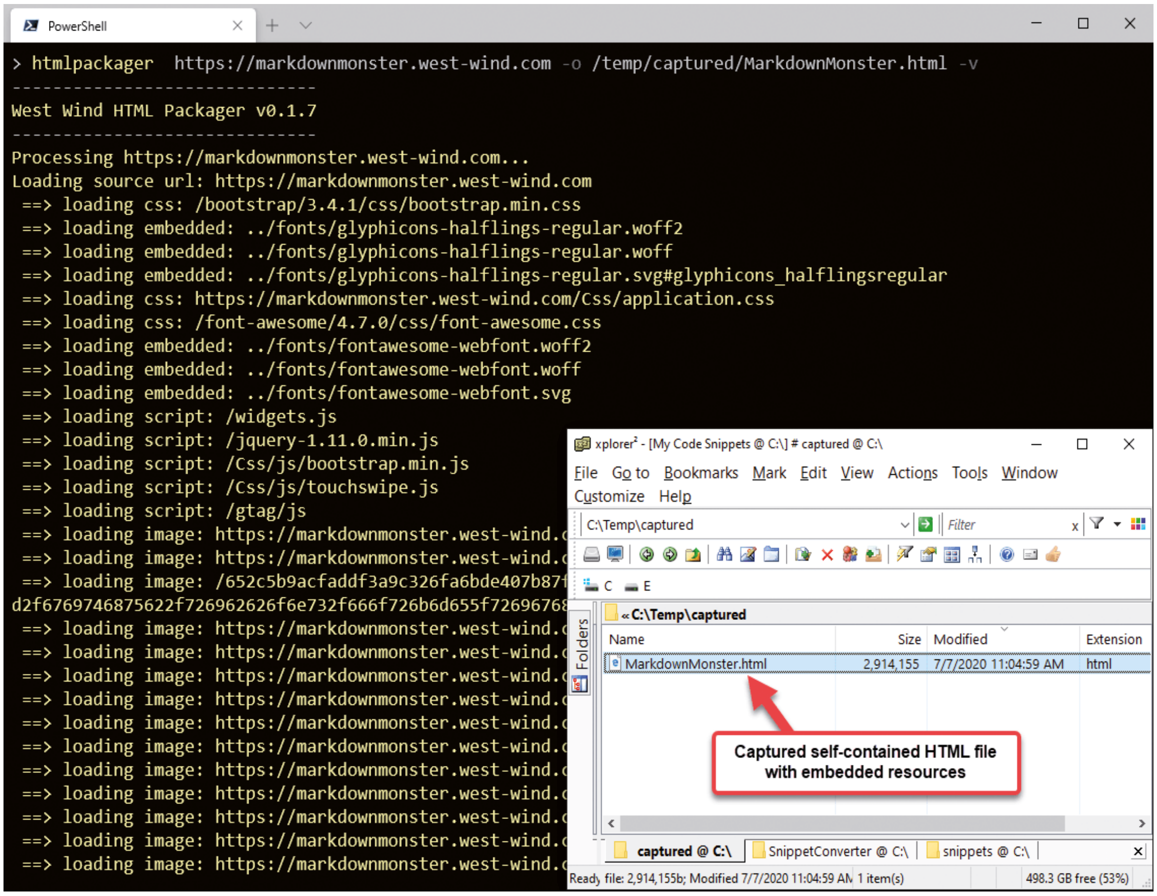
```
dotnet tool install -g LiveReloadServer
```

To run it, you simply point it at a **--WebRoot** folder on your local computer and that folder then becomes the content source for the website. **Figure 13** shows the LiveReload server running with requests firing through it.

The Live Reload functionality is enabled by default so you can edit static text files and immediately see the changes reflected in active browser pages. If you have Razor Pages and/or Markdown Page processing enabled, you can edit Razor and Markdown pages and see those pages refreshed immediately as well.

Because LiveReloadServer is a .NET Core Tool, it also works great on Mac and Linux even though I originally built it for my local Windows set up. After making some very minor adjustments for pathing for Mac and Linux support, the server now also works on those platforms.

By default, the server runs as a local computer Web server, but it can also expose an external port, so it can also be used to host externally accessible sites. This is great for giving network access to other users on your internal network, or for applications that need to expose Web server functionality. The server binaries can also be hosted as full Web applications in IIS or other external front end servers like NGINX, because it's essentially an ASP.NET Core-hosted application.



```
> htmlpackager  https://markdownmonster.west-wind.com -o /temp/captured/MarkdownMonster.html -v
------------------------------
West Wind HTML Packager v0.1.7
------------------------------
Processing https://markdownmonster.west-wind.com...
Loading source url: https://markdownmonster.west-wind.com
 ==> loading css: /bootstrap/3.4.1/css/bootstrap.min.css
 ==> loading embedded: ../fonts/glyphicons-halflings-regular.woff2
 ==> loading embedded: ../fonts/glyphicons-halflings-regular.woff
 ==> loading embedded: ../fonts/glyphicons-halflings-regular.svg#glyphicons_halflingsregular
 ==> loading css: https://markdownmonster.west-wind.com/Css/application.css
 ==> loading css: /font-awesome/4.7.0/css/font-awesome.css
 ==> loading embedded: ../fonts/fontawesome-webfont.woff2
 ==> loading embedded: ../fonts/fontawesome-webfont.woff
 ==> loading embedded: ../fonts/fontawesome-webfont.svg
 ==> loading script: /widgets.js
 ==> loading script: /jquery-1.11.0.min.js
 ==> loading script: /Css/js/bootstrap.min.js
 ==> loading script: /Css/js/touchswipe.js
 ==> loading script: /gtag/js
 ==> loading image: https://markdownmonster.west-wind.
 ==> loading image: https://markdownmonster.west-wind.
 ==> loading image: /652c5b9acfaddf3a9c326fa6bde407b87
d2f6769746875622f726962626f6e732f666f726b6d655f72696768
 ==> loading image: https://markdownmonster.west-wind.
 ==> loading image: https://markdownmonster.west-wind.
 ==> loading image: https://markdownmonster.west-wind.
 ==> loading image: https://markdownmonster.west-wind.
 ==> loading image: https://markdownmonster.west-wind.
 ==> loading image: https://markdownmonster.west-wind.
 ==> loading image: https://markdownmonster.west-wind.
 ==> loading image: https://markdownmonster.west-wind.
 ==> loading image: https://markdownmonster.west-wind.
 ==> loading image: https://markdownmonster.west-wind.
 ==> loading image: https://markdownmonster.west-wind.
```

xplorer² - [My Code Snippets @ C:\] # captured @ C:\

C:\Temp\captured

Name	Size	Modified	Extension
MarkdownMonster.html	2,914,155	7/7/2020 11:04:59 AM	html

Ready file: 2,914,155b; Modified 7/7/2020 11:04:59 AM 1 item(s) 498.3 GB free (53%)

Figure 12: Running HTML Packager against a URL produces a single self-contained HTML file.

LiveReloadServer is a simple, generic local Web server you can point at a local folder to host a website.

A Legacy Web and Application Server

The final example is for a legacy application server running as a .NET Core Tool. It's for an ancient integration tool that has recently been updated to .NET Core as an option. I maintain a FoxPro legacy product called **West Wind Web Connection** that still has many users—for nearly 25 years now. This is as legacy as it comes, and you may laugh at technology this old, but it's still in use.

Because the tools I created so many years ago extended FoxPro functionality to build Web applications, there are still a lot of existing applications that use large FoxPro code bases, haven't been updated, and just keep ticking along. These tools have been kept current and continue to allow these ancient applications to keep working without major disruption or requiring complete re-writes on other platforms for which there is often no budget, technical skill, or incentive.

Web Connection has, for many years, used .NET technologies to provide the interface between a Web Server (IIS only in the past) and FoxPro and the tech using .NET and COM (ugh) is surprisingly stable and reliable. So, when .NET Core came out a few years ago, it offered several new opportunities to extend the Web Server interface tooling Web Connection implements by:

- Providing a fully self-contained development server
- Providing a nearly zero configuration environment
- Its ability to run the server component on non-Windows computers

.NET Core Tool Security

.NET Core Tools are essentially executable programs that are downloaded and executed on your local computer. There's no review process for uploaded components, so there's potential for misusing this technology to install malicious software.

Always make sure you trust the publisher of a tool. If that's not possible, check for an open source repository to review the source code and the issues around the project to verify that the project is legitimate and secure.

Practice safe computing—better safe than sorry!

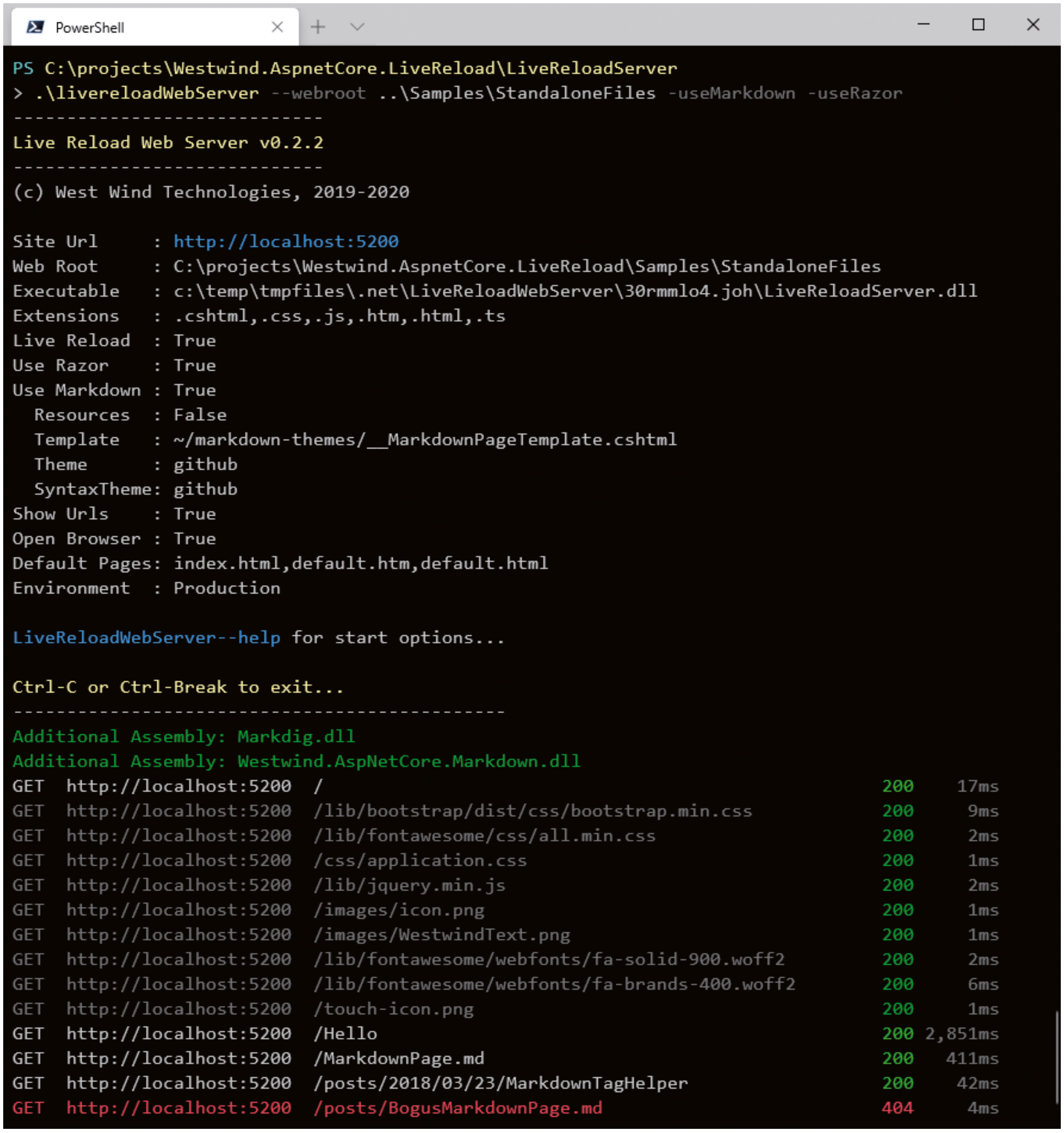
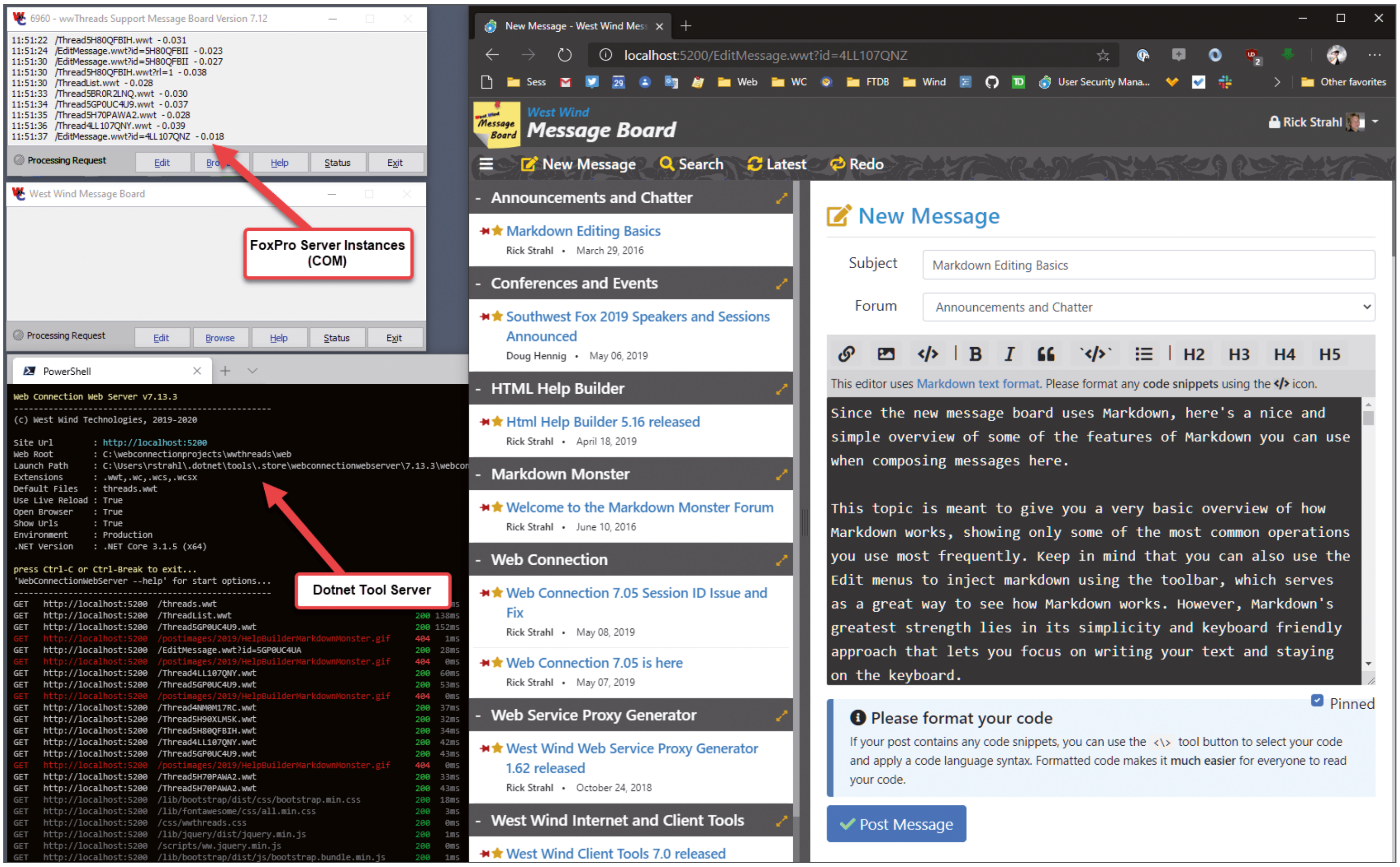

```
PS C:\projects\Westwind.AspnetCore.LiveReload\LiveReloadServer
> .\livereloadWebServer --webroot ..\Samples\StandaloneFiles -useMarkdown -useRazor
-----------------------------
Live Reload Web Server v0.2.2
-----------------------------
(c) West Wind Technologies, 2019-2020

Site Url     : http://localhost:5200
Web Root     : C:\projects\Westwind.AspnetCore.LiveReload\Samples\StandaloneFiles
Executable   : c:\temp\tmpfiles\.net\LiveReloadWebServer\30rmmlo4.joh\LiveReloadServer.dll
Extensions   : .cshtml,.css,.js,.htm,.html,.ts
Live Reload  : True
Use Razor    : True
Use Markdown : True
  Resources  : False
  Template   : ~/markdown-themes/__MarkdownPageTemplate.cshtml
  Theme      : github
  SyntaxTheme: github
Show Urls    : True
Open Browser : True
Default Pages: index.html,default.htm,default.html
Environment  : Production

LiveReloadWebServer--help for start options...

Ctrl-C or Ctrl-Break to exit...
----------------------------------------------
Additional Assembly: Markdig.dll
Additional Assembly: Westwind.AspNetCore.Markdown.dll
GET  http://localhost:5200  /                                               200    17ms
GET  http://localhost:5200  /lib/bootstrap/dist/css/bootstrap.min.css       200     9ms
GET  http://localhost:5200  /lib/fontawesome/css/all.min.css                200     2ms
GET  http://localhost:5200  /css/application.css                            200     1ms
GET  http://localhost:5200  /lib/jquery.min.js                              200     2ms
GET  http://localhost:5200  /images/icon.png                                200     1ms
GET  http://localhost:5200  /images/WestwindText.png                        200     1ms
GET  http://localhost:5200  /lib/fontawesome/webfonts/fa-solid-900.woff2    200     2ms
GET  http://localhost:5200  /lib/fontawesome/webfonts/fa-brands-400.woff2   200     6ms
GET  http://localhost:5200  /touch-icon.png                                 200     1ms
GET  http://localhost:5200  /Hello                                          200 2,851ms
GET  http://localhost:5200  /MarkdownPage.md                                200   411ms
GET  http://localhost:5200  /posts/2018/03/23/MarkdownTagHelper             200    42ms
GET  http://localhost:5200  /posts/BogusMarkdownPage.md                     404     4ms
```

Figure 13: LiveReloadServer generically serving Web content out of a local folder.



```
11:51:22  /Thread5H80QFBIH.wwt  - 0.031
11:51:24  /EditMessage.wwt?id=5H80QFBII  - 0.023
11:51:30  /EditMessage.wwt?id=5H80QFBII  - 0.027
11:51:30  /Thread5H80QFBIH.wwt?rl=1  - 0.038
11:51:30  /ThreadList.wwt  - 0.028
11:51:33  /Thread5BR0R2LNQ.wwt  - 0.030
11:51:34  /Thread5GP0UC4U9.wwt  - 0.037
11:51:35  /Thread5H70PAWA2.wwt  - 0.028
11:51:36  /Thread4LL107QNY.wwt  - 0.039
11:51:37  /EditMessage.wwt?id=4LL107QNZ  - 0.018
```

```
Web Connection Web Server v7.13.3
---------------------------------------------------
(c) West Wind Technologies, 2019-2020

Site Url        : http://localhost:5200
Web Root        : C:\webconnectionprojects\wwthreads\web
Launch Path     : C:\Users\rstrahl\.dotnet\tools\.store\webconnectionwebserver\7.13.3\webcon
Extensions      : .wwt,.wc,.wcs,.wcsx
Default Files   : threads.wwt
Use Live Reload : True
Open Browser    : True
Show Urls       : True
Environment     : Production
.NET Version    : .NET Core 3.1.5 (x64)

press Ctrl-C or Ctrl-Break to exit...
'WebConnectionWebServer --help' for start options...
---------------------------------------------------
GET   http://localhost:5200  /threads.wwt
GET   http://localhost:5200  /ThreadList.wwt                                   200 138ms
GET   http://localhost:5200  /Thread5GP0UC4U9.wwt                              200 152ms
GET   http://localhost:5200  /postimages/2019/HelpBuilderMarkdownMonster.gif   404   1ms
GET   http://localhost:5200  /EditMessage.wwt?id=5GP0UC4UA                     200  28ms
GET   http://localhost:5200  /postimages/2019/HelpBuilderMarkdownMonster.gif   404   0ms
GET   http://localhost:5200  /Thread4LL107QNY.wwt                              200  60ms
GET   http://localhost:5200  /Thread5GP0UC4U9.wwt                              200  53ms
GET   http://localhost:5200  /postimages/2019/HelpBuilderMarkdownMonster.gif   404   0ms
GET   http://localhost:5200  /Thread4NM0M17RC.wwt                              200  37ms
GET   http://localhost:5200  /Thread5H90XLM5K.wwt                              200  32ms
GET   http://localhost:5200  /Thread5H80QFBIH.wwt                              200  34ms
GET   http://localhost:5200  /Thread4LL107QNY.wwt                              200  42ms
GET   http://localhost:5200  /Thread5GP0UC4U9.wwt                              200  43ms
GET   http://localhost:5200  /postimages/2019/HelpBuilderMarkdownMonster.gif   404   0ms
GET   http://localhost:5200  /Thread5H70PAWA2.wwt                              200  33ms
GET   http://localhost:5200  /Thread5H70PAWA2.wwt                              200  43ms
GET   http://localhost:5200  /lib/bootstrap/dist/css/bootstrap.min.css         200  18ms
GET   http://localhost:5200  /lib/fontawesome/css/all.min.css                  200   3ms
GET   http://localhost:5200  /css/wwthreads.css                                200   0ms
GET   http://localhost:5200  /lib/jquery/dist/jquery.min.js                    200   1ms
GET   http://localhost:5200  /scripts/ww.jquery.min.js                         200   0ms
GET   http://localhost:5200  /lib/bootstrap/dist/js/bootstrap.bundle.min.js    200   1ms
```

- Announcements and Chatter
  - Markdown Editing Basics — Rick Strahl • March 29, 2016
- Conferences and Events
  - Southwest Fox 2019 Speakers and Sessions Announced — Doug Hennig • May 06, 2019
- HTML Help Builder
  - Html Help Builder 5.16 released — Rick Strahl • April 18, 2019
- Markdown Monster
  - Welcome to the Markdown Monster Forum — Rick Strahl • June 10, 2016
- Web Connection
  - Web Connection 7.05 Session ID Issue and Fix — Rick Strahl • May 08, 2019
  - Web Connection 7.05 is here — Rick Strahl • May 07, 2019
- Web Service Proxy Generator
  - West Wind Web Service Proxy Generator 1.62 released — Rick Strahl • October 24, 2018
- West Wind Internet and Client Tools
  - West Wind Client Tools 7.0 released

New Message

Subject: Markdown Editing Basics

Forum: Announcements and Chatter

This editor uses Markdown text format. Please format any code snippets using the </> icon.

```
Since the new message board uses Markdown, here's a nice and
simple overview of some of the features of Markdown you can use
when composing messages here.

This topic is meant to give you a very basic overview of how
Markdown works, showing only some of the most common operations
you use most frequently. Keep in mind that you can also use the
Edit menus to inject markdown using the toolbar, which serves
as a great way to see how Markdown works. However, Markdown's
greatest strength lies in its simplicity and keyboard friendly
approach that lets you focus on writing your text and staying
on the keyboard.
```

☑ Pinned

Please format your code

If your post contains any code snippets, you can use the <\> tool button to select your code and apply a code language syntax. Formatted code makes it **much easier** for everyone to read your code.

Figure 14: WebConnectionWebServer running a FoxPro server application through .NET Core.

- Its ability to run a local computer or network server without requiring IIS
- Its ability to run truly local Web applications like desktop applications

Long story short, a .NET Core Tool turned out to be great delivery mechanism for someone who needs to run an old application that was built with Web Connection, but doesn't have access or can't install IIS on a computer and most of all, who doesn't want to set up and configure IIS.

This sounds strange, but it's a very common scenario: A user calls and says they have a Web Connection application that was developed 15 years ago, the developer left, and they need to run the application and perhaps make a few changes. Setting up an environment in the past wasn't the easiest thing in the world. With this new component, I can whittle that down to:

- Install the .NET Core SDK.
- Install **dotnet tool install -g WebConnection.**
- Point at a Web Connection folder and go.

A .NET Core Tool here offers a smooth solution to distribute a server that can then be used to serve an ancient legacy application without complex configuration or special installations required. On the local computer, it's literally: point at a Web Connection application folder and go—no further configuration is required. The same server implementation can also be deployed in a live environment with IIS.

The internal server implementation is very similar to the Live Reload server but extended to handle the Web Connection server protocols so it can handle Web Connection-specific script handling and request routing that lets FoxPro server code execute in response to Web requests. The .NET Core implementation was moved from a .NET HTTP Handler to .NET Core Middleware, and it was surprisingly easy with 95% code reuse from the existing handler.

Legacy technology is always an eye roller, and I doubt that any of you will use this technology, but it makes for an interesting use case of the .NET Core Tool functionality in a very unconventional scenario.

Summary

.NET Core Tools provide a great way to share executable code from .NET Core in an easy and platform-independent way. If you've built a NuGet package and published it before, you already know how to build and share a .NET Core Tool. .NET Core Tools are easy to consume, which gives access to a variety of existing tools and, because they're so easy to share, promote a community of tools to be created.

Hopefully, this article has given you some ideas of tools that you might want to use, or better yet, create and share your own with the community or even just within your organization.

Rick Strahl

CODE

SPONSORED SIDEBAR:

Get .NET Core Help for Free

Looking to create a new .NET Core project? Get off on the right foot with a FREE hour-long CODE Consulting session. No strings. No commitment. Just CODE. Our consultants have been working with the .NET Core and ASP.NET Core teams since the early pre-release builds. For more information, visit www.codemag.com/consulting or email us at info@codemag.com.

Talk to an RD

Dr. Neil Roodyn and Markus Egger

In this "Talk to an RD" column, Markus Egger and Dr. Neil Roodyn continue their virtual conversation about the impacts of the COVID-19 crisis. Both are involved with managing teams that are now working remotely more than before. Both are involved with customers facing the same issues. This discussion focuses on security issues.

Markus: Do you use Surface Hubs in your organization?

Dr. Neil: I've been using Surface Hubs, but I suspect they're all switched off right now since nobody is in the office. But I'm very, very fortunate. I have a Surface Studio on my desk, which is a beautiful device. A lot of the teams that I work with have Surface tablets, Surface Pro, or Surface Books. Or a Lenovo tablet that has a pen or an iPad with a pen. Microsoft Whiteboard runs on any Windows device and on iPad. If you have a pen, you get the extra bit of nice functionality where you can engage. I think the other nice thing about Whiteboard is that, as different people are scribbling on the same whiteboard at the same time, you see their little icons popping up all over the screen so you can track who's doing what.

You can see a bigger picture emerge as multiple people are scribbling down. You can divide the page into columns and people can enter stuff in their own columns, or however you want to do it. You can create notes and move them around on the page. I found that to be a very useful free-form tool in the same way I've always liked having a whiteboard in a working space when I'm in the same room as someone. It encourages that wider conversation. I think sharing screens fluidly and being able to share screens among different people in the conversation is also very powerful. People want to share, whether it's a new UI they're designing or they want to have a conversation around some plan they've put together. Just being able to do that in Teams has been powerful.

Bringing in guests is also very powerful. Knowing that you can lock MS Teams down to people within your organization, but you can have teams where you can bring in guests in a controlled fashion is another aspect of it. There are lots of pieces of the puzzle that connect together to make a very useful toolkit that MS Teams has presented.

Markus: I believe you can use the Whiteboard product whether you use Teams or not, is that correct?

Dr. Neil: Yes. The Whiteboard product isn't attached to Teams. It's a separate product. It's attached to Active Directory. Most of the organizations I work with are enterprise-scale organizations. They have their own Office365, so we're working with those on the inside of their firewall, logged in as an AD under their domain.

Markus: Overall security is pretty good with all that stuff, right? I mean, looking at Teams and the security that it has compared to some competitor products that have made big headlines lately with various security problems. It's always nice to know that in Teams, security works well. Also, most people already have a license for it because it's part of Office.

Dr. Neil: Right. Microsoft made a big move to make it free for everyone during this period. So that's made it useful too, in that if I have a team that's not necessarily hooked into the Microsoft ecosystem, I can add them as guests into my domain and bring them into conversations.

The security aspect, certainly within a domain within an organization, is incredibly important. Because there's a lot of these conversations you don't want getting outside of the firewall, right? In the situation we're in due to COVID-19, we need to be able to have senior leadership conversations that you normally would have behind closed doors in a boardroom. You need to do them virtually. You need to be sure that they're secure. You want to have an environment that you're 100% comfortable in.

And there are other aspects too, like knowing that something is recorded. You can see a little dot on your screen right now because I'm recording this call. It's also important knowing that something has been captured or isn't being captured. It's just extra transparency and extra information.

Markus: I know you're also very interested in approaching software development with security in mind and as a driving factor.

Dr. Neil: It's what I've started calling "Security Driven Development," copycatting the Test-Driven Development that I started pushing 20 years ago when I was doing all the XP [eXtreme Programming] drive, trying to get people into XP and Agile, and thinking about how to put quality first. One of the things that surprises me a little bit in the last

The Microsoft Regional Director Program

The Regional Director Program, or RDs for short, is a program that allows Microsoft to identify influential individuals in an effort to give the community-at-large access to these individuals, and also to provide a point of communication and feedback into Microsoft. Regional Directors do NOT work for Microsoft (and they aren't paid for being part of the RD program), but they have a formal relationship with Microsoft that provides them with great insights and connections within Microsoft.

The Microsoft Regional Director website (https://rd.microsoft.com) defines the RD program in the following words:

"The Regional Director Program provides Microsoft leaders with the customer insights and real-world voices it needs to continue empowering developers and IT professionals with the world's most innovative and impactful tools, services, and solutions. Established in 1993, the program consists of 160 of the world's top technology visionaries chosen specifically for their proven cross-platform expertise, community leadership, and commitment to business results. You will typically find Regional Directors keynoting at top industry events, leading community groups and local initiatives, running technology-focused companies, or consulting on and implementing the latest breakthrough within a multinational corporation."

Regional Directors are expected to have deep technical understanding of many of the Microsoft technologies. Not just that, but RDs are expected to have an understanding of, and experience with, competing technologies. RDs are also expected to go beyond technical expertise and have considerable business expertise. Many RDs present at corporate events, advise governments and NGOs, and many similar scenarios.

Feel free to contact any of the Regional Directors to get access to an RD's expertise and a well-informed opinion that isn't shaped or influenced by having to go along with Microsoft's marketing speak. You can contact RDs to get advice and opinions on your projects regarding technical needs. You can contact them to help analyze how technology will influence your business. You can invite RDs to speak to your project stakeholders, board of directors, or corporate event. And you can contact your RD for many more scenarios.

year or so was how many developers don't really think about security stuff. They're very feature-focused and want to deal with security later.

The problem with dealing with security later is the same problem you have with dealing with quality later. You can build something that's very hard to secure. If you build something test-driven, it's very hard to build something that's not high quality because you're building it so it's testable. If you build something security-driven, it's very hard to build something that's not secure because you're building it with security in the forefront of your mind.

There's a lot of this kind of change of attitude that needs to happen. Especially as more developers are building greater scaled applications that offer services. You think of the architectures we're being encouraged to move our world into, of microservices, or treating the cloud as just a place to host functions that you can call. These are all great ideas, but you have to make sure that you're not breaking security. "Oh yeah. We've got this new service that, you know, exposes all this customer information that you can call." That's is great internally. And then you wonder who else can call it. "Oh, anybody." And that's not good. [laughs]

Well, let's lock that down. You have to think about that from the beginning. You want to make sure that there's good authentication and validation. I've been doing a bunch of work helping people understand. None of it's really that complicated. But you must understand how to make sure that they're using best practices from the start and not going "oh, well, we'll use HTTPS to start with, and then we'll retrofit some other security stuff on top of that." It's not breaking news, but HTTPS isn't super secure. And if you're passing information that's critical to your business, maybe you want to think about not just encrypting it yourself, but also validating the source and the destination and other places you're expecting them to be. You can do that with certificates, with signing, and a whole bunch of tech that's already out there. It's not very hard to get your head around, but you have to plan to do that from the beginning.

Markus: Do you have any pointers? Are there any books or articles people should read or anything like that?

Dr. Neil: I think there are a few things and it really depends on what your platform is and where you're going. The great thing about this time that we live in is that security is front and center for all the big platforms. So whether you're on Amazon and AWS, or you're on Microsoft and Azure, or you're on Google, they all have great documentation on the security aspects of their platform and how to make sure the products that you're building on their platforms are secure. You really don't have to dig very hard to find that documentation. Whatever platform you're on, go and find out what the security best practices are and read them, and make sure you understand what they're trying to tell you, because none of them want to be the platform that exposes, you know, seven million customer records because you did something silly. [laughs] These platforms have a vested interest in supporting your efforts to be secure.

The second thing is that, again, in this day and age, there are a huge number of online courses that are free or very cheap. You can go to LinkedIn Learning, or Pluralsight, and you can find a ton of really good security courses for your language and for your platform. If you're programming in Java, C#, or C++. It really doesn't matter. You'll find security training in that language. If you're building stuff for IoT on little microchips and shipping 10 million of them around the world, you'll find a course that teaches you how to secure those devices. There's no shortage of information.

To some degree, that's why I was a little bit shocked to start discovering that a large number of developers weren't really thinking about security first. They were stuck on this cycle of features. I understand why; it's because the business is driving them to build features. The reality is that they have to think about security first and that was crossing all kinds of industries that I've stuck my finger into in the last decade—from the finance markets through to logistics and shipping

In this feature, we eavesdrop on a conversation between Dr. Neil Roodyn and Markus Egger, both seasoned Regional Directors.

Talk to **Dr. Neil** about:
Software development, cloud computing, More Personal Computing, cognitive services, bots, conversation as a platform, true social computing, robotics, quantum computing, the economic singularity, and the future.

Talk to **Markus** about:
machine learning and AI, ASP.NET, HTML apps (including Angular, Vue.js, etc.), the cloud, Azure, Microservices, Windows applications, software strategy, and much more.

You can contact him at markus@eps-software.com

Dr. Neil Roodyn has long had a passion for software development, going back to the 70s when he taught himself BASIC and 6502 Assembler. In the 90s, Neil worked on a number of different real-time systems that led to a thesis entitled "Software Architectures for Distributed Real-Time Systems." Neil was awarded a PhD for this thesis from University College London and the brand of Dr. Neil was born. Since 1995, Dr. Neil has been involved in the formation of many software companies, his roles varying from mentoring through to directorship. Dr. Neil keeps a very active involvement in the software being produced and delivered today, and provides feedback to both large and small companies on how to increase the value of their software business.

Markus Egger is not just an RD, but as the founder and publisher of CODE Magazine, he's also directly associated with this publication. In his main job, he is the President and Chief Software Architect of EPS Software Corp. (a company better known for its CODE brands such as CODE Consulting, CODE Training, CODE Staffing, and CODE Magazine). He is also the founder of other business ventures, such as Tower 48, Wikinome, and others. In his own words, he spends his time "like most software developers, writing production code" both for consulting and custom app-dev customers, and also for his own companies. He has worked for some of the largest companies, including many Fortune 500 companies, such as Microsoft. Markus often takes on the role as a "CTO for hire."

through to property management. It's everywhere. I think if your goal is to build a product that is a global-scale success, if you're trying to build a product that's gonna make you the next Apple, Microsoft, Google, Facebook, whatever, you have to think about these things. And it's true even if you're building an internal product.

I guess the other thing that's really brought a lot of this to the fore, is horrific legislative activities, like GDPR. Suddenly everyone's going "The what? What do we have to do? What do you mean we can't sell our product in Europe anymore?!?" In many ways, for as much as I hate all the bureaucracy, I think it's done everyone a big favor in making them think about what personal information is. What is this data that I'm transferring among 50 different servers? What does it mean? How am I transferring it? Is it accessible to other people? I think there are a number of factors that have come into play. The whole GDPR conversation is what's also driven a lot of these security courses online.

Markus: How much does moving onto one of those clouds help you with that? Some of the cloud stuff forces you into that, doesn't it? You put up a SQL Azure database and you don't have quite the freedom to mess up security as you did before. But it's not automatic by any stretch of imagination.

Dr. Neil: No. And you can break it. [laughs] There are some defaults that all the clouds, whether it's Amazon or Microsoft or whoever, give you that are definitely more secure than you would be than if you had just gone and installed SQL Server on a box and connected it to an Internet connection and started serving up data to your website. There are some defaults that they'll set up and there are also some flags that they all give you. They give you a little flag saying "Warning. We don't believe this is secure. You should not use this technique. Using this other technique is our recommendation." Not that they actually prevent you from doing it wrong. You can go and configure it how you want and it could be insecure. But the flag is there. And every time you log into their portal, you'll get this alert: "Your SQL server is not considered to be secure the way we would like it, or the way we would recommend it here."`

Markus: Or they might even send you an email.

Dr. Neil: Yeah. "Click this link to see our recommendations." There is definitely a base level of value you get from that. But then you get to code you've written and you've pulled that data out of the database, and now you're going to manipulate it somehow, and you're going to send it somewhere else, you're kind of out of the realm of them worrying about it. If you're using their messages, the cloud messaging protocol that they've provided, maybe you're secured again by what they're providing. But I think the other thing is to make sure you understand what they're providing. If you're using some messaging system hosted on a server somewhere hosted by Amazon or Microsoft, understand what they're providing as part of that. Even just the fact that you've gone and read and understood, means you're now getting a bit more of your head around the concept of what security means.

When they say something that you don't understand, go and look it up! Hopefully you then understand a bit more about that aspect of the security. I always recommend to everyone that they investigate what's currently provided, even if they're using a standard cloud platform provider. But the other thing I often say is "don't trust it." You want to make sure that it's secure as far as you're concerned or not only secure as far as they're concerned. And if you want to be the one who's comfortable with security, maybe you need to add something to that. Maybe you need to add a level of encryption or a level of validation. If the only server that's allowed to send you this kind of information is this or that server, sign it or have a certificate that's on that server. Then you know it came from the right server and it can't come from somewhere else.

Markus: I'd imagine you're still an advocate of not home growing your added solution. You go with certain standards and encryption mechanisms and standard signing and so on.

Dr. Neil: Absolutely! I don't want to invent a new security protocol. Absolutely not. At the same time, I don't think a lot of what comes out of the box is necessarily the best thing. Like, I think a lot of the certificate signing is still 2048 [bit encryption], and really, we probably need to go up to 4096 now. You may just want to flip that switch so that you're using a slightly harder-to-crack set of tokens. But yeah, I do think that you should first understand it and then use the tools to make sure you're making the security harder to crack and harder for someone to access the data.

Markus: How do you deal with sign-on and IDs and accounts and all that?

Dr. Neil: [laughs] That's a super interesting conversation because identity is core to everything we're doing. One of the things I spent a bunch of time on last year is looking at different identity solutions. Also digging into this concept of Decentralized Identity. Right now, you think about identity for everything we do. Let's say you log onto Teams; you log on with your username password, and maybe you use multifactor authentication or two factor authentication and get a thing that comes up on your phone and you say, "yes," and you're in and that's great. But who owns that identity? Well, now it's a Microsoft identity, or it's an Active Directory identity, or you log into Gmail and it's now a Google-owned identity, or you log into your Amazon admin portal and you're now on an Amazon identity, or you log into your Adobe tools and you're now on an Adobe identity.

It's like, "wait a minute!" How many identities did I just roll off? Like five, six? And you have all of those, right? I'm sure you do. I'm sure most people have this huge number of identities. Oh, and you also have a government identity. When you log onto your driving license or to pay your taxes, that's a different identity. So you just start thinking "wait a minute, there's something really wrong with this picture!" Well, there are multiple things wrong with this picture [laughs], but the first one is that most of them are still relying on a username and password. And 20 years of doing this makes me feel like this really isn't right anymore. The other thing is that you're now giving ownership of the identity of who you are, in different aspects of it, to different people, not yourself. You no longer own your own identity in the Adobe cloud. You no longer own your own identity in the Amazon cloud.

The compliant way of thinking about it is blockchain or ledger-based identities.

Could we change it? Can we turn the whole thing on its head and create a world where you own your identity and you decide which parts of it you want to give to different people? You then say, "oh, well, for my health records, obviously, I want this part of my identity, like my date of birth, my address, my next of kin." Whatever. Those kinds of things. That's part of what you share with your health identity, but you still log on as the same person when you go to Adobe. They don't need to know anything about any of that stuff, right? Maybe they need to know your billing address, but they don't need to know your next of kin or your date of birth.

You might share a different set of components, but it's the same identity. That's this concept of trying to create a decentralized identity solution. The buzzword-compliant way of talking about it is blockchain or ledger-based identities. That's a technique that you could use to create this decentralized identity, but it definitely seems like an interesting way forward and a way out of this multi-identity problem and this lack of consolidation to who you really are. The great thing is that if you were to do this, you could start getting crossover. Say, for example, I want to buy a domain and get a certificate on Azure. Well, how do they know that it's really me? At that point, I could share my government identity with them with the same identity that I'm already logged in with and say, "here's my driving license and here's my passport number and you're allowed to use it for the next two days in order to issue me with my own domain and my own certificate for that domain." And then it disappears from their data. I'm giving permission to use this for a period of time and then it gets revoked.

These kinds of concepts are technically very feasible now. I'm keen to try to work out how I can start helping organizations think about using a more decentralized identity within their environment. Because I think that this would enable a much richer digital conversation to start happening between systems.

Markus: So how do we all imagine this? We were talking about things like blockchain, but I'm assuming we're not just talking about blockchain-style technology that sits on a server, but we would have a very large distributed set up. Almost like Bitcoin using blockchain?

Dr. Neil: Yeah. It has to be a set of distributed ledgers that capture the different aspects of what we're all doing.

Markus: Blockchain would mean it's trustworthy. It's not changed and so on. It's not fake. But how do you give it authority in the first place?

Dr. Neil: You'd have authorities in the ledger. The Australian government or Hawaiian state government are examples of such authorities. Or you may have the department of driving, or you may have hospitals with certain authorities. You can specify what authorities they have in order to be able to carry out certain actions. Or to be able to add something to the chain or to your ledger saying, "this is Markus and yes, this is his driving license as issued by Hawaiian driving."

Markus: There would be other entities that would have the ability to add or allow you to add to it and sign it in some way to know it's actually for real.

Dr. Neil: Yes. And you could do all of this with certificates, as well. Let's say that I have an certificate issued to me and Markus says "oh, I trust what Neil says about a member of staff who used to work with Neil" and I've signed something and validated that the person was a good developer and worked with me and I really enjoyed it and I've signed it. There are a lot of these kinds of things that start to become super interesting. We'd have to get to a point where we could enable people to really carry identities in a digital form. Right now, that probably means on their phones. You'd carry your identity with you wherever you go and be able to share different aspects of it on different systems with different people.

Markus: Do you envision that will be a completely new blockchain, or could it reuse something like an Ethereum blockchain?

Dr. Neil: There are lots of different projects afoot right now. And if anyone's interested, they can go and look them up, but there is one based on Ethereum. There's one based on what you might call the "Bitcoin blockchain." Microsoft has a research project called "Ion" that's kicked off.

I think what needs to happen, to be of real value, is it needs to be disconnected from coin. The problem with the ones that are connected to coin, in my view, is that then it becomes a pay-for-use scenario. Obviously, they're motivated by coin-spending. In order to validate someone to sign something, or to do something of that kind of activity, you pay a price. I'm not sure that's correct. I don't necessarily believe you can scale that. If you were going to have hundreds of millions of transactions happening every minute on the Internet, does it makes sense that there's a coin value attached to every single time someone logs onto a website to do online shopping? I think that the future of it needs to be disconnected from any coin-based blockchain and needs to be independent. There are a few already starting to pop up that are doing this.

Markus: It makes total sense. There's no reason why a blockchain would need to be coupled to a coin.

Dr. Neil: The ones that have coin were quite quick to adopt it because they already have the infrastructure.

Markus: You could just use an Ethereum-based blockchain for this.

Dr. Neil: Exactly. You don't need to have coin attached to it, but of course the platforms require payment because they're a business of some sort. They're looking for ways to gain revenue out of this.

Markus: Who would be the clearing house for that? Or who provides the API so companies like Adobe could integrate into that? If you're talking about entities like Microsoft or Google, now we're almost back to a blockchain-based version of Microsoft Passport.

Dr. Neil: I think there are two aspects to it. You just brought up two completely different terms. One was "clearing house" and then "APIs." I think the whole point of this is that there's no clearing house anymore. Everybody acts as their own clearing house to a certain degree. You validate what you want to have shared from your ledger with other people. You are the person who owns your identity. The API is a different aspect. I'm sure that over time, this would evolve and there will be Microsoft APIs and Google APIs and Amazon APIs for ledger-based identity or certificate-based identity or however it ends up being mushed together. But I think the concept of a clearing house is what disappears as part of this.

> I'm not convinced that we're taking the right route in trying to lock everything down.

I think there's a whole other aspect of this identity security conversation. It's probably controversial, but I'm not convinced that we're taking the right route in trying to lock everything down. The concept of privacy in the modern world is quite a modern concept. If you think about us as ancient human beings living in small tribes, there wasn't really a concept of privacy. I'd know exactly who you were sleeping with and what you were eating, because there are only 37 of us in the tribe. [laughs] The concept of privacy in a tribal world is kind of bizarre, actually. A lot of cultures never even contemplated privacy as a thing that was a right or even desirable. Everyone knew what everyone else was up to and you knew straight away that the guy over there is a jerk because it was obvious because everything was out in the open.

You still see this a little bit in cultures that have small communities. I was kind of shocked about this the first time I went to countries like Finland. I went to Helsinki and was amazed at just how safe it was. I used to joke that in a park in Helsinki, you could put your laptop down, go for a walk for 10 minutes, get your coffee, walk back, and your laptop would still be there. I remember once asking a Finnish person about it and she said, "yeah, absolutely." Because if so-and-so steals your laptop, everyone in the whole town knows it was him by tomorrow.

Anonymity removes accountability and the online world has pushed that to the extreme.

Markus: Anonymity removes accountability and the online world has pushed that to the extreme.

Dr. Neil: Yeah. The online world has created this extra level of anonymous behavior and anonymous activity. And maybe the solution is to get rid of all of that and make everything incredibly transparent.

Markus: But then doesn't that go counter to your security-driven development?

Dr. Neil: Absolutely! [laughs] That's why I think it's interesting! Because I think there are two ends to this scale. You either go to everything being totally locked down or you go to nothing being locked down. It's the in-between that we're stuck in that's so troublesome.

Markus: Then perhaps we should put everyone's behavior into a blockchain. You can look up all the bad stuff someone did. You know, Big Brother to the max!

Dr. Neil: You're not the first to suggest this. [laughs] Some people would say this is highly restrictive. You're now observing my every behavior. What are you doing wrong that you don't want to be observed? And you have to ask: "why do you need to be anonymous if you're not taking bad action?" Are you embarrassed by your activities? You don't want to let people know that you went to that website. Well, why did you go to their website? Maybe you should think about that a little bit more and not go to that website. Let's, for the sake of the conversation, say it's a gambling website. I know that wasn't what you were thinking. [laughs] So maybe you shouldn't have gone to that gambling website. If you're embarrassed about going to that gambling website, you should have thought about that a little bit harder.

Markus: That's true. On the other hand, it could also be a completely different matter. Maybe you're a person in the public eye and you don't want everyone to know about your family life. You take your kids to a certain school and you don't necessarily want everybody to know where your kids are at all points in time in order to protect them. Then it takes on a little bit of a different dimension.

Dr. Neil: It does. Except if you know where all the stalkers are all the time, then maybe that's less of a problem. [laughs]

What I'm saying is that I think if you really want to solve this, you either go all the way to absolute maximum security. Everything is locked down and nobody really knows who anybody is. Or you go all the way to complete transparency, where there is nothing hidden at all. Then if I want to know where someone was at some point in time, it's there. It's the in-between that we have the problems with, where some people are more anonymous than others. And some people's activity is more hidden than other people's activity. I think that's where a lot of these challenges start. Well, how is he allowed to be secret about it? I'm not secret about it.

If you are an amateur money launderer, shall we say,...

I think just the same with money laundering. Like if you're an amateur money launder, shall we say. And I know this because I used to work...

Markus:
...as a money launderer? [laughs]

Dr. Neil: [laughs].

Markus: Where do we go with this? We may be going somewhere that you don't want the world to know. [laughs]

Dr. Neil: [laughs].I used to work in financial surveillance. I used to look for bad patterns of behavior and movements of money. There are some interesting things that you learn when you do that. One of them is that people who know what they're doing are never going to get caught. The people who generally get caught for doing silly things with not paying taxes or whatever, are people who don't understand what the rules are or don't understand in general. So what I'm really saying is if you're good at hiding it, you can hide it.

That means it's different rules for different people. What I'm saying is if you went to complete transparency, you break all that down. We're dealing with this right now. We're dealing with it here in Australia and you've been dealing with it in the US. We have the same problem here in Australia where we have a very high percentage of indigenous people locked up in prisons compared to the overall percentage of the population. There's clearly something broken with that. I think that's partly because of the lack of transparency in the whole system. You can look at this almost throughout the entire world. In some places it's a little less so and somewhere it's a little more so, but we're certainly seeing something that's a systemic problem worldwide and hopefully we're seeing a change. Something will have to change there.

Markus: That's actually a very interesting philosophical discussion.

Dr. Neil: Yes! I am not sure how fitting this is for an RD column, but it's super interesting. The society that we live in globally is built on the history that made it how it is. We have to accept that part of history involved some pretty horrible things. Like slavery, like genocide, like the capture of countries that didn't belong to the conquerors. Let's face it: The British empire was created because they had guns and the other people didn't. We have to really understand how we got to this point before we can come to solutions.

This goes all the way back to Roman society. It's not hugely surprising that a lot of legal systems around the world are based, at some level, on Roman law. That's because the Romans worked out legal mechanisms that justified their ability to build an empire, take over countries, capture people, turn them into slaves, give them mechanisms to become free, have earnouts. These are all concepts that are intrinsic to our behavior and our society. So yes, this is not an easy "let's turn this switch over to six and we'll be fixed" type of situation.

Markus: This is a pretty heavy conversation for a coding magazine.

Dr. Neil: (laughs) It is. It's something that needs to be discussed everywhere. Nothing can change until we see what's happening, whether it's in our code or our society.

Markus: True. Thanks for meeting with me. I'll be seeing you online or at a conference sometime.

Dr. Neil: It was fun. Thanks!

Markus Egger

Blockchain: A Practical Application

This article is a practical application of Wei-Meng Lee's May/June 2018 article: Understanding Blockchain: A Beginner's Guide to Ethereum Smart Contract Programming (https://bit.ly/3i2fu2C). Of all the most talked about and hyped topics in technology today, blockchain is at the top of the list. Almost without exception, the terms blockchain and crypto currency are used interchangeably for the basic reason that crypto currencies are based on blockchain. But not all blockchains are crypto currencies. It follows that we can discuss and implement blockchain independently of crypto currencies and the peer-to-peer networks within which the blockchains that are central to crypto currencies reside.

John V. Petersen
johnvpetersen@gmail.com
linkedin.com/in/johnvpetersen

Based near Philadelphia, Pennsylvania, John is an attorney, information technology developer, consultant, and author.

None of that resolves the basic question: What are blocks and block chains? What does one look like? Assuming there's some value with them, does blockchain, as a concept, apply to your applications? If so, which problems can be solved with blockchain and how would you implement those solutions in your applications? To answer those questions, you need a tangible proof of concept that makes concrete what has previously been largely an abstract concept. In this article, I tackle these questions with a tangible blockchain proof-of-concept.

Although this article endorses blockchain and despite referring to the crypto currency context, it's not an endorsement of crypto currencies. If there's one thing to take away from this article, it's that although blockchain and crypto currencies are related, they are in fact, quite distinct things. Crypto currencies are still in their infancy and have had their fair share of problems. Whether crypto currencies succeed or fail, that has no bearing on blockchain's efficacy. Editorially speaking, despite the hype, I don't see crypto currencies as a viable long-term thing. That conclusion is based on several factors:

- Acquisition difficulty (at least in the USA)
- Association with nefarious activity
- Uncertain recourse when something goes wrong
- Not immune to theft
- General uncertainty and risk concerns

With a few strokes of the legislative pen, crypto currencies as a form of anonymous payment could very well be outlawed. Who are the only people that place a premium on anonymity in financial transactions? Terrorists, money launders, drug dealers, and sex traffickers, to name a few. No legitimate activity requires crypto currency. Despite the issues with crypto currency, those issues should not and do not detract from the benefits of blockchain, the primary enabler of crypto currency.

The code for the proof-of-concept discussed in this article can be found here: https://github.com/johnvpetersen/BlockChain. Note: because this is an active repository, although conceptually consistent, there are some deviations with the implementation details. It's baked enough to discuss, but this project is very much a work in progress. The goal is for you to take what I've started and make it useful for your specific use cases.

A Quick Word on Encryption

For the purposes of this proof-of-concept and article, data at rest isn't encrypted, but it could be. Encryption, however, is a separate concern. If data within a block is encrypted, it's the responsibility of another facility to decrypt and read the block data. Perhaps that would call for an IBlockReader interface? Perhaps that interface has a generic method that accepts a JSON string that was generated from the block's ToString() method?

What Is a Blockchain?

Blockchain isn't a technology or a third-party service for which you need to purchase a seat. You may have heard the phrases "distributed digital ledger" and "smart contracts" in the context of decentralization and crypto currencies like Bitcoin. Blockchain is the foundation of distributed digital ledgers and smart contracts. Distributed digital ledgers have been compared to a spreadsheet that's replicated and synced via nodes that are connected via a peer-to-peer network. A smart contract is a program that manages the protocol coordinating actions that are incident to some legal obligation. An example of such an obligation is the requirement to pay a sum of money in return for a product or service. Blockchain is at the core of it all. The specific programs that implement blockchain and the network those programs reside on, those are things I'm going to put to the side so that I can focus on blockchain itself as a thing you can implement in your applications. To do that, you need to understand what a blockchain is and what problems it's well suited to solve.

As glib as it may sound, it's nevertheless accurate to say that a blockchain is a chain of blocks. That necessarily raises two questions: What are blocks and how are blocks chained?

As **Figure 1** illustrates, a block is a data container. A block includes at least two things:

- Data
- Data Hash

The block data is composed of the following items:

- Business data (order, customer, transaction, or anything else)
- The previous block's hash. If this is the first block, the value is null
- A nonce value (random integer)

A block's validity is determined by calculating the hash and then comparing that value to the stored hash. If those values are equal, the block is valid. In a blockchain, illustrated in **Figure 2**, a block's previous hash value must equal the preceding block's hash value. The only exception to the rule is the first block, or what is often referred to as the "Genesis block." The first block's previous hash value will always be null. Carrying the previous hash forward to the next block is how the blocks are chained together.

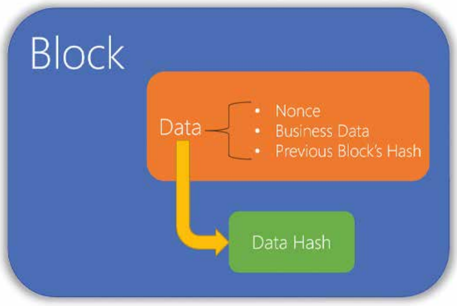


Figure 1: A block is a data container that also includes the hash of its data.

What Is the Nonce Value, What Is It Used For, and How Does It Relate to Proof-of-Work?

In cryptography, a nonce value is an arbitrary value that's used to generate a hash. In a block, the business data and previous hash values are fixed. The generated hash value for those two things will always be the same. What if the generated hash needs to conform to some specification such as the hash needing to begin with a set prefix? The only way to generate a different hash value is to update the nonce value.

A common way to generate nonce values is via a random number generator. This implies a process that continually loops through a process that generates a new random number, generates the hash, and then compares the generated hash value to a required prefix specification. This process is the proof-of-work concept that's also a core concept in the crypto currency context. Proof of work is the way nodes on a network achieve consensus on whether a block can be added to the chain. Another phrase you may have heard of is "block mining." The more complex the specification for the hash, the more computing power you need to find the first combination of elements that yields a hash that conforms to the specification so that the block can be added to the chain.

For an internal application, do you need to implement the proof-of-work concept? The answer is no. Nevertheless, I've implemented the concept here because if there's at least some rule for hash complexity and that rule is secret and embedded in your program, it follows that your program—and only your program—can create blocks. In such a case, a block's provenance is certain, which in turn, enhances the blockchain's data trustworthiness. In other words, a rogue process can't create or update blocks in a blockchain.

A related concept to proof-of-work is proof-of-stake. With proof-of-work (PoW), any miner that expends the necessary effort (including energy consumption) can verify a new block. Because of the computing resources required, the energy consumption mentioned is just that: electrical power. Miners get rewards for mining blocks. Those rewards fund the sizeable electrical bills that are incurred with block mining! Proof-of-stake (PoS), on the other hand, goes toward how a subset of miners acquire the right to verify new blocks. This dichotomy between PoW and PoS is one example that illustrates the open questions related to crypto currencies.

Blockchain Benefits

Because a block's data contains the hash for the previous block, this arrangement has at least two profound ramifications:

- The validity of any one block can be verified. Does the generated hash == the stored hash? We can easily detect whether a block's data has been altered since the hash was generated.
- The validity of the blockchain can be verified. Does a block's data contain the hash for the previous block?

You often have the need to quickly verify whether a data series is "Valid." Valid, in this context, means that the data hasn't been altered since hashes were generated. Altering any one block in the chain not only invalidates that block, but it also invalidates every subsequent block and, as a result, the entire chain is invalidated. If, for a given data set, you were called upon to prove whether the dataset has changed, how would you go about handling that task? Is it possible? Would you need to ping off another data source? If you implement blockchain, the task is trivial because a blockchain is capable of expressing whether or not data has been altered since hashes were generated and it can do so within its boundaries without the need for external resources. This can have profound positive performance ramifications for your applications! Therefore, in the crypto currency context, blockchain supports the decentralized digital ledger concept.

Blockchain Use Case: Read-Only Sequential Data

The world isn't a nail and blockchain isn't a hammer! For blockchain, a good use case appears to be one that involves read-only sequential data. Think about bank account transactions. Once a transaction has been logged, it should never be altered. Once an event occurs and has been recorded, there's no going back. In other words, there is no Wayback Machine that allows you to go back in time to change history. If a correction needs to be made, you simply append a new transaction to record the correction. Or put another way, add another block to the chain.

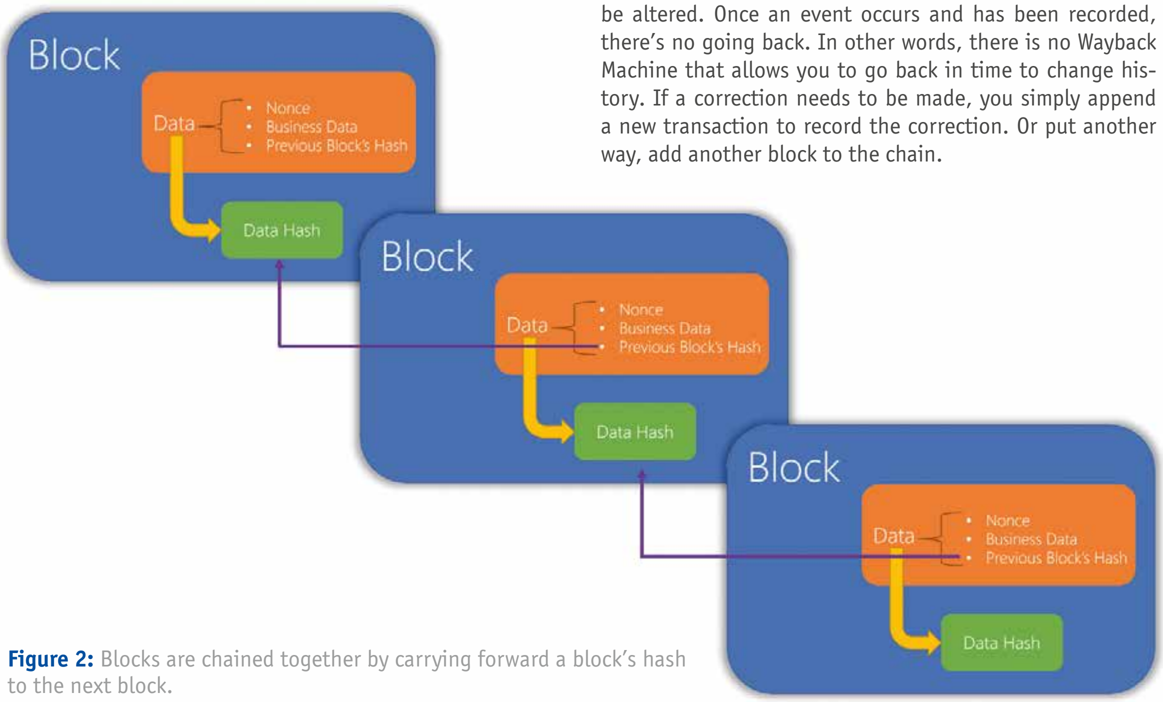


Figure 2: Blocks are chained together by carrying forward a block's hash to the next block.

Blockchain applies to any scenario that involves the recordation of an event. Blocks are always added to the end of the chain. In the crypto currency world, this gets a bit more complicated because multiple copies of the blockchain exist on nodes in the network. This is what the distributed digital ledger is all about. Synching up the nodes is the job of a network service like Ethereum. For the purpose of this article and this use case, you don't need to be concerned with a third-party network or the need to synch-up distributed nodes. All you need to be concerned with is with blockchain itself.

With that background, let's review some code!

Implementing a Blockchain in .NET Core

You can find the project code in the following GitHub Repository: https://github.com/johnvpetersen/BlockChain. To fully understand the code, I strongly encourage you to run and step through the tests. Although I used VS Code and .NET Core 3.1, you can easily offload the code to Visual Studio and leverage .NET 4.x if you're more familiar with that environment. The solution, illustrated in **Figure 3** is divided between two projects:

- **App:** hosts the core blockchain classes
- **Tests:** hosts the test fixtures that exercise the blockchain classes hosted in the app

As **Figure 4** illustrates, a block chain is a simple structure. The way the blocks are chained is also quite simple. The link is a block's hash that's carried forward to the next block's data.

To implement what's depicted in **Figure 4**, you'll rely on the following objects:

- **Data:** An object to hold a block's data. The data structure includes the nonce, business data, and if applicable, the previous block's hash.
- **Proof of work:** An object that uses a specified algorithm to calculate the required prefix for a valid hash.
- **Block:** An object to host data. The block object is also responsible for calculating its data hash.
- **Chain:** An object to host a collection of blocks. The chain object is responsible for adding a new block to the collection. Part of that operation includes applying the previous block's hash to the new block's data.

Data Class

Listing 1 illustrates the data class implementation. Because the class is generic, it can host your custom class. In addition to your business data, this class also contains the nonce value and the previous block's hash. The following illustrates how to instantiate and verify the data class:

```
[Fact]
public void CanInstantiateData() {
var sut = new Data<MyData>(
   0,new MyData(10,"Amount is $10.00"));
Assert.Equal(10,sut.Value.Amount);
Assert.Equal("Amount is $10.00",
  sut.Value.Message);
Assert.Equal(0,sut.Nonce);
Assert.True(
  string.
  IsNullOrEmpty(sut.PreviousHash));
}
```

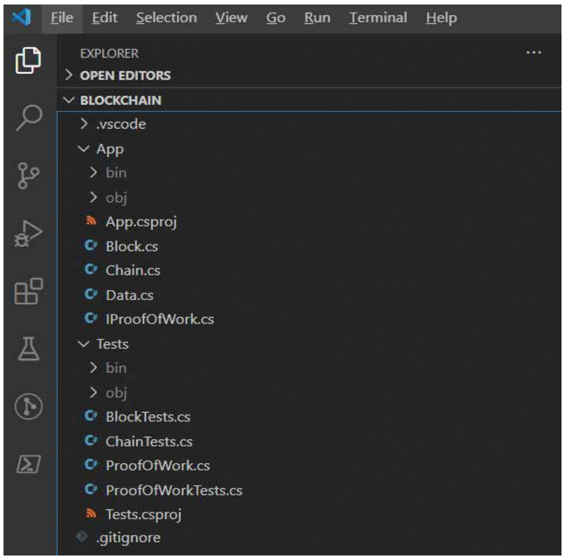
Overriding the ToString() Virtual Method

For each class, take note of how the ToString() method is always overridden. In each case, a class can emit itself as JSON when its ToString() method is invoked. The purpose is to facilitate serialization and de-serialization. All of the classes in this solution are immutable once they've been instantiated. During its lifetime, a blockchain will move from active to inactive states. When inactive, the blockchain only exists as JSON. When active, that JSON string is used to instantiate the blockchain class. In the examples that follow, you will see how the overridden ToString() method is useful.

IProofOfWork Interface

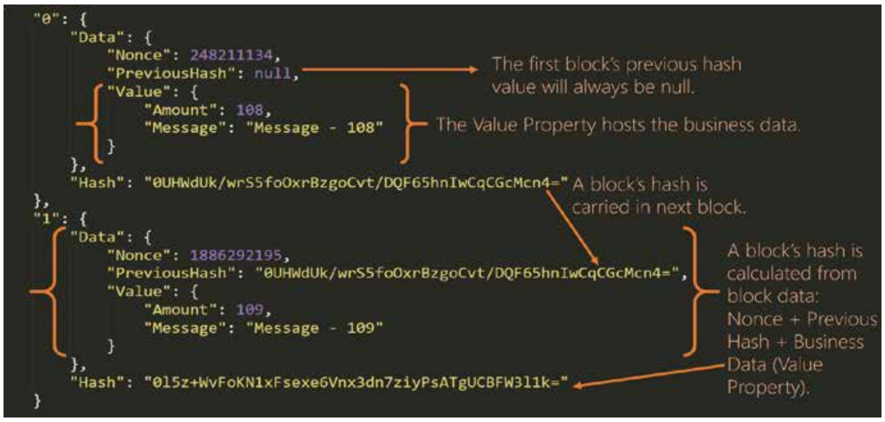
The solution defines an interface, not an implementation for proof-of-work. The reason for the interface is that you don't know at design time what specific rule will be employed to define the required prefix for a hash. The following illustrates this simple interface with one method:

```
public interface IProofOfWork {
      string GetPrefix();
   }
```



```
> .vscode
v App
  > bin
  > obj
  App.csproj
  Block.cs
  Chain.cs
  Data.cs
  IProofOfWork.cs
v Tests
  > bin
  > obj
  BlockTests.cs
  ChainTests.cs
  ProofOfWork.cs
  ProofOfWorkTests.cs
  Tests.csproj
.gitignore
```

Figure 3: The blockchain implementation is very simple in that it only consists of three classes and one interface.



```
"0": {
    "Data": {
        "Nonce": 248211134,
        "PreviousHash": null,
        "Value": {
            "Amount": 108,
            "Message": "Message - 108"
        }
    },
    "Hash": "0UHWdUk/wrS5foOxrBzgoCvt/DQF65hnIwCqCGcMcn4="
},
"1": {
    "Data": {
        "Nonce": 1886292195,
        "PreviousHash": "0UHWdUk/wrS5foOxrBzgoCvt/DQF65hnIwCqCGcMcn4=",
        "Value": {
            "Amount": 109,
            "Message": "Message - 109"
        }
    },
    "Hash": "015z+WvFoKN1xFsexe6Vnx3dn7ziyPsATgUCBFW3l1k="
}
```

The first block's previous hash value will always be null.
The Value Property hosts the business data.
A block's hash is carried in next block.
A block's hash is calculated from block data: Nonce + Previous Hash + Business Data (Value Property).

Figure 4: A block chain is a chain of blocks where each block contains data and the data hash. The data is an object that contains a nonce, business data, and, if applicable, the previous block's hash.

Listing 1: Data Class

```
using static Newtonsoft.Json.JsonConvert;

namespace BlockChain {
   public class Data<T> {
      public Data (int nonce, T value, string previousHash = "") {
         Nonce = nonce;
         Value = value;
         PreviousHash = previousHash;
      }
      public int Nonce { get; private set; }
      public T Value { get; private set; }
      public string PreviousHash { get; private set; }

      public override string ToString () {
         return SerializeObject (this);
      }
    }
}
```

Listing 2: Proof Of Work Class

```
public class ProofOfWork: IProofOfWork
{
 private string _prefix;

 [JsonConstructor]
 public ProofOfWork(string prefix) {
 _prefix = prefix;
}

 public ProofOfWork() : this(Guid.NewGuid(),new int[] {0}) {}

 public ProofOfWork(Guid guid, int[] charsToTake) {

 _prefix = string.Empty;
 if (charsToTake == null || charsToTake.Length == 0)
  return;

 Array.ForEach(charsToTake, element => {
  _prefix += guid.ToString().Substring(element,1);
 });
}

 public string GetPrefix() => _prefix;
}
```

Listing 2 illustrates the implementation of the IProofOfWork interface. The class can be instantiated with a set prefix or with a GUID and an array of integers that specifies which characters in the GUID to use for the prefix. **Note: the longer the prefix, the longer it will take the generate a hash.** The following test is an example of how to implement the proof of work class that implements the IProofOfWork interface:

```
[Fact]
public void CanVerifyPrefixWithParameters () {
 var guid = Guid.NewGuid ();
 int[] charsToTake = { 0, 5, 10 };
 var expected = string.Empty;

 var sut = new ProofOfWork (guid, charsToTake);

 Array.ForEach (charsToTake, element => {
  expected +=
   guid.ToString ().Substring (element, 1);
 });

 Assert.Equal (expected, sut.GetPrefix ());
}
```

The implementation above is completely arbitrary. The class's only purpose is to encapsulate code that generates a prefix. That code can be as simple or as complicated as you need it to be. It's also worth noting that IProofOfWork implicates all the SOLID principles:

- **S: Single Responsibility:** IProofOfWork supports concrete implementations doing one thing and one thing only. That one thing is to generate the required hash prefix.
- **O: Open-Closed:** Because the dependency is on an abstract contract, the entity relying on the abstract contract is open for extension. Extending capability is achieved by way of the concrete class that conforms to the IProofOfWork Interface.
- **L: Liskov Substitution:** An entity relying on the IProofOfWork Interface can work with any implementation of that interface.
- **I: Interface Segregation:** In this context, you can alter how a required hash prefix is calculated without affecting any other aspect of the solution.
- **D: Dependency Inversion:** In this context, the chain and block objects depend on the IProofOfWork interface, not on any specific concrete implementation. At runtime, the chosen concrete implementation that conforms to the interface can be injected into the chain and block classes.

Block Class

The block class joins the data and proof-of-work concepts. To review, the block is a data container and is illustrated back in **Figure 1**. The block class code is illustrated in **Listing 3**. Like the other classes, the block class overrides the ToString() method such that the ToString() method leverages JSON.NET's object serialization capability. The serialized block object, as is illustrated in **Figure 4**, corresponds to the block constructor that's decorated with the JsonConstructor attribute. This is how you can serialize and de-serialize a block. It's important to emphasize that all classes in this solution are immutable. You need a way to hydrate an object while it's in an active state. When inactive and at rest, you need a way to persist state.

JSON serialization and deserialization is how to achieve that capability and the following test illustrates how to create, serialize, and deserialize a block. The MyData class referenced in the following test is the business object that hosts the business data that's ultimately hosted in the block. The block class has a private method to compute the data hash. Could that functionality be offloaded to its own class and then have the class instance (object) injected into the block? The answer is an unqualified "yes" and would be a worthwhile refactoring exercise, one that I've left outstanding for you to undertake if you wish. The same is true for how a block generates a hash. As built, that specific functionality is burned into the block class definition. If you want to be able to extend that functionality, that requires the code to be refactored to its own class and then you'd rely on dependency injection to make that functionality available with the block class.

```
public void
  BlockCanBeSerializedAndDeSerialized () {
 var sut =
 new BlockChain.Block<MyData> (
  new MyData (108, "Amount is 108"),
                  null,
                  new ProofOfWork ()
              );
  var blockJSON = sut.ToString ();
  sut = null;
  sut =
  DeserializeObject<BlockChain.Block<MyData>> (
   blockJSON
  );

  Assert.True (sut.IsValid ());
}
```

Chain Class

The chain class code is illustrated in **Listing 4.** The two public members to focus on are the Blocks Property and the AddBlock() Method:

- **Blocks Property:** This property is the public version of the internal blocks dictionary. The System Collections Immutable NuGet Package is leveraged so that you can create an immutable version of the internal blocks dictionary. An external entity needs to read the block data in the chain. At the same time, the data needs to be read-only. Although the underlying block objects and the data it contains is read-only, the outer dictionary is mutable. It isn't a concern because the variable that holds that mutable dictionary (_blockChain) is private. The internal dictionary needs to be mutable because blocks may be added. When you refer to the Blocks property, a copy of the _blockChain variable is made. That copy is, however, an immutable dictionary.
- **AddBlock() Method:** The AddBlock() method is responsible for taking the block data, taking the block data and computing a hash, and checking to see if the computed hash conforms to the prefix if a required prefix applies. That's why the While loop is in place. The process continues to leverage the block's ability to generate hashes until one is generated that conforms to the prefix. To achieve that capability, the nonce value is updated with another random number. If that didn't happen and given that the other data doesn't change, the same hash value is generated and unless the hash generated conforms to the prefix, you'd never exit the loop. In this context, much of this operation occurs in the block's constructor. In other words, when a block is created, it necessarily means the block's hash conforms to the required prefix if one was specified.

The following is a test that puts everything together. There are two broad scenarios tested: with a proof of work prefix requirement and without. The test, under both scenarios:

- Creates a new chain
- Adds three blocks
- Serializes the chain
- Nulls the blocks variable
- Deserializes the chain from JSON
- Adds a fourth block

```
[Theory]
[InlineData(true)]
[InlineData(false)]
public void
ChainCanBeSerializedAndDeSerialized (
   bool proofOfWork
) {
  var blocks =
  new Chain<MyData> (proofOfWork ?
  new ProofOfWork() : null);
  blocks.AddBlock (
   new MyData (108, "Amount is 108"));
  blocks.AddBlock (
   new MyData (109, "Amount is 109"));
  blocks.AddBlock (
   new MyData (110, "Amount is 110"));
```

Listing 3: Block Class

```
using System;
using System.Security.Cryptography;
using System.Text;
using Newtonsoft.Json;
using static Newtonsoft.Json.JsonConvert;

namespace BlockChain
{
 public class Block<T> {

 private string _hash = string.Empty;
 private Data<T> _data;
 private string _prefix;

 [JsonConstructor]
 public Block (string hash, Data<T> data,string prefix) {
  _hash = hash;
  _data = data;
  _prefix = prefix;
 }

 public Block (
  T data, string previousHash = "",
  IProofOfWork proofOfWork = null) {
   _prefix =
    proofOfWork == null ?
      string.Empty : proofOfWork.GetPrefix ();
  var rnd = new Random ();
  while (true) {
   var blockData = new Data<T> (rnd.Next (), data, previousHash);
   var result = computeHash(blockData);

   if (string.IsNullOrEmpty (_prefix) ||
    result.Substring (0, _prefix.Length) == _prefix) {
    _hash = result;
    _data = blockData;
      break;
     }
    }
  }
 string computeHash () => computeHash (_data);

 string computeHash(Data<T> data)
 {
  using (var sha256 = SHA256.Create())
   {
    return Convert.ToBase64String(
     sha256
      .ComputeHash(
       Encoding.UTF8.GetBytes(
        SerializeObject(
         data
        )
       )
      )
     );
    }
   }
 public override string ToString () {
  return SerializeObject (this);
 }
 public string Hash => _hash;
 public Data<T> Data => _data;
 public bool IsValid () {
  var retVal = true;
  retVal = string.IsNullOrEmpty (_prefix) ? retVal :
   this.Hash.Substring (0, _prefix.Length) == _prefix;

   return retVal && computeHash (_data) == this.Hash;
  }
 }
}
```

Listing 4: Chain Class

```
using System.Collections.Generic;
using System.Collections.Immutable;
using Newtonsoft.Json;
using static Newtonsoft.Json.JsonConvert;
using System.Linq;

namespace BlockChain {

public class Chain<T>
 {

private Dictionary<int, Block<T>> _blockChain =
 new Dictionary<int, Block<T>>();
 private IProofOfWork _proofOfWork = null;
 public ImmutableDictionary<int, Block<T>> Blocks =>
  _blockChain.ToImmutableDictionary();

[JsonConstructor]
public Chain(
 Dictionary<int, Block<T>> blockChain,
 IProofOfWork proofOfWork)
 {
  _proofOfWork = proofOfWork;
  _blockChain = blockChain;
 }

 public Chain(IProofOfWork proofOfWork = null)
 {
  _proofOfWork = proofOfWork;
 }

 public bool? IsValid()
 {
  if (Blocks == null || Blocks.Count == 0)
   return null;
  if (Blocks.Count(x => !x.Value.IsValid()) > 0)
   return false;
  for (int i = 1; i < Blocks.Count; i++)
  {
   if (Blocks[i].Data.PreviousHash != Blocks[i - 1].Hash)
    return false;
  }
  return true;
 }

 public override string ToString()
 {

 var prefix =
 _proofOfWork == null ? string.Empty : _proofOfWork.GetPrefix();

 return SerializeObject(
  new
  {
   BlockChain = _blockChain,
   ProofOfWork = prefix
  }
 );
}

public void AddBlock(T data)
{
 var previousHash =
 _blockChain.Count == 0 ?
 null :
 _blockChain[_blockChain.Count - 1].Hash;
 _blockChain.Add(
  _blockChain.Count,
  new Block<T>(
   data,
   previousHash,
   _proofOfWork)
  );
 }
 public Block<T> this[int i] => _blockChain[i];
}
}
```

```
Assert.Equal(3,blocks.Blocks.Count);

Assert.True (blocks.IsValid ());

var blockJSON = blocks.ToString ();

blocks = null;

blocks =
DeserializeObject<Chain<MyData>> (
  blockJSON, new ProofOfWorkConverter());

Assert.True (blocks.IsValid ());
Assert.Equal(blockJSON, blocks.ToString());

blocks.AddBlock (
 new MyData (111, "Amount is 111"));

Assert.Equal(4,blocks.Blocks.Count);
      }
  }
```

Conclusion

Blockchain, independent of the crypto currency context, is a simple and powerful concept. A prime use case for blockchain in a business application is one where there are multiple sequentially created records. The blockchain itself can express whether it's in a valid state. A valid state is one where the computed hash equals the newly generated hash for the data. Assuming nothing changed with the data, the computed and stored hashes are identical. In the crypto currency world, there's the concept of block mining, which involves significant computing power to create a block, Block mining entails the proof-of-work concept. The way the proof of work concept is implemented in this article is through the proof of work class that defines a required prefix. When a block is created with a proof of work requirement, the process of looping and determining whether the new hash conforms to the prefix, which is conceptually the same as block mining in the crypto currency world. **Note: if you require a generated hash to conform to a prefix length that is greater than or equal to 3, you will likely see significant delays when generating blocks. It may very well be that in a business application, there is no need for proof-of-work. That's why the proof-of-work was separated to its own interface.**

Finally, the chain class ties the blocks together. Like a block class, the chain class can express its valid state. If any one block is invalid, the entire chain is also invalid. Being able to determine at a glance and without having to ping off another resource like a database whether or not a dataset is valid is a performance enhancement. Although you can't prevent data tampering with 100% success, you can, at the very least, with blockchain, detect when data has been altered. In my opinion, that ability makes blockchain a powerful approach that affords your applications benefits that would otherwise be more difficult to realize. It's important to note that hashing as a means to detect data modifications isn't a new or novel concept, and neither is blockchain!

John V. Petersen

The Complete Guide to Suspense in Vue3

Suspense is a new feature that will see the light in the long-awaited release of Vue 3. It's highly inspired by React Suspense (https://reactjs.org/docs/concurrent-mode-suspense.html). This article will introduce Suspense, showcase the various ways of using it, show it in action with example app code, and end with ways to handle errors.

What's Suspense in Vue3?

While researching and learning about Suspense, I read many articles and watched many videos. I had light bulb moments and moments of confusion. It's the confusion that compelled me to explain it clearly and with my own examples.

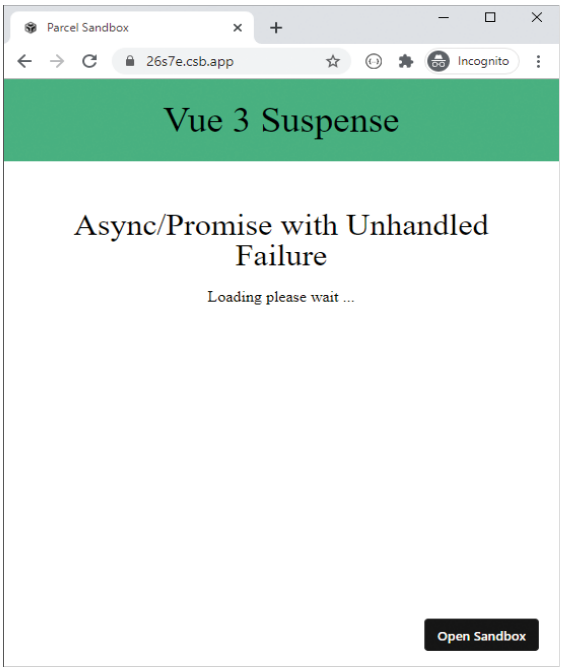
Suspense creates an execution context or a boundary around the content it wraps. It waits for the component(s) wrapped inside to be ready before displaying it/them. Meanwhile, it displays a fallback content that could be a text message, spinning animation, or any other type of content.

Initially, the Suspense component displays the fallback content by default. The wrapped component makes use of the async **setup()** function and awaits an async/promise operation to fetch data from a back-end server. The moment the promise is resolved successfully, the Suspense component displays the component on the screen.

The Composition API in Vue 3 is implemented via the setup() function. To learn more about the Composition API, check out my article, "Vue 3 Composition API, Do You Really Need It?" (https://labs.thisdot.co/blog/vue-3-composition-api-do-you-really-need-it).

If, for whatever reason, the async/promise operation fails or is pending, the Suspense component continues to show the fallback content. You'll see this in the coming sections.

Let me illustrate this with a few examples. Let's assume that you have a component that's wrapped inside a Suspense component that uses the Composition API in Vue 3 by using the setup() function.


Parcel Sandbox

26s7e.csb.app

Vue 3 Suspense

Async/Promise with Unhandled Failure

Loading please wait ...

Open Sandbox

Figure 1: Async/Promise with Success.

Async/Promise Operation with Success

In this first scenario, I'm going to execute an async/promise operation that results in a successful operation. **Listing 1** shows the complete source code for this section.

You can play with this example on Suspense with Success here: https://codesandbox.io/s/suspense-success-rzwx3.

The fetch() function, part of the Fetch API (https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API), communicates with a remote REST API to return COVID-19 data by country. It returns a Promise.

The fetchData() function simulates a delay by using both a Promise and the setTimeout() function.

The moment the Promise is resolved and the data is available, the SampleOne component is ready to be rendered. At this time, the wrapping Suspense component inside the App component displays it and hides the fallback content. **Figure 1** shows the app running.

This example demonstrates the safe path. The Suspense component deals with a successful async/promise operation with no surprises!

Async/Promise Operation with Unhandled Failure

In this second scenario, I'll execute an async/promise operation that results in an unhandled failure. To simulate a Promise rejection, I'll use an incorrect host name in the request URL and see how it behaves.

```
async function fetchData() {
  return new Promise(async (resolve) => {
    const res = await fetch(
      `
        https://disease.sh/v3/covid-19/countries/
        ?yesterday=true&strict=true
      `
    );
    setTimeout(async () =>
        resolve(await res.json()), 2000
    );
  });
}
```

Check out this example on Suspense with Unhandled Failure: https://codesandbox.io/s/suspense-unhandled-failure-26s7e.

The rest of the code is kept the same. Notice the use of **diseases** instead of **disease**.

Needless to say, the request fails and the Promise returned is rejected. How does the Suspense component deal with this?

Bilal Haidar
bhaidar@gmail.com
https://www.bhaidar.dev
@bhaidar

Bilal Haidar is an accomplished author, Microsoft MVP of 10 years, ASP.NET Insider, and has been writing for CODE Magazine since 2007.

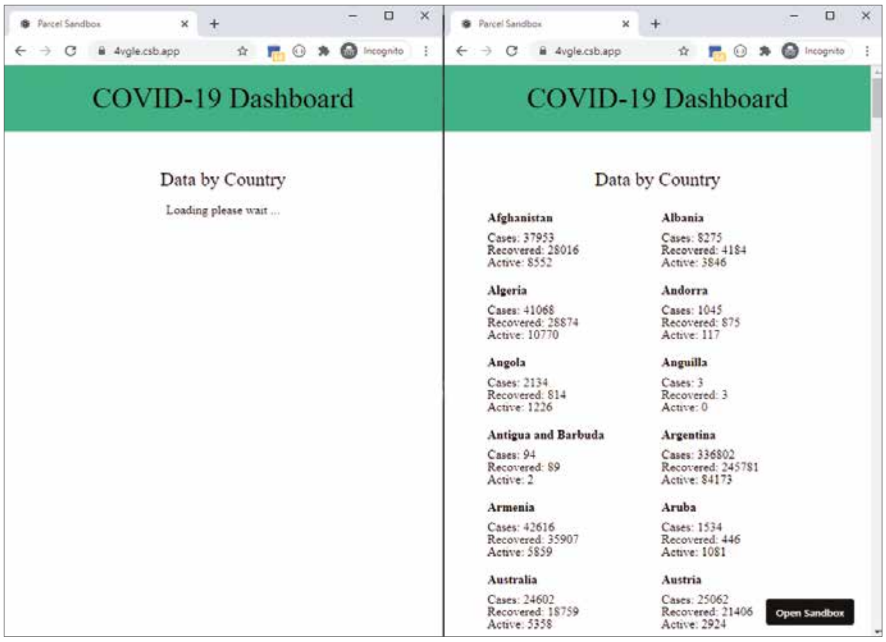
With 15 years of extensive experience in Web development, Bilal is an expert in providing enterprise Web solutions.

He works at Consolidated Contractors Company in Athens, Greece as a full-stack senior developer.

Bilal offers technical consultancy for a variety of technologies including Nest JS, Angular, Vue JS, JavaScript and TypeScript.

Listing 1: Async/Promise Operation with Success

```
import { createApp, ref } from "./vue.esm-browser";

const SampleOne = {
  name: "Sample One",
  async setup() {
    const data = ref(null);

    data.value = await fetchData();

    return {
      data
    };
  },
  template: `
    <pre>{{ data }}</pre>
  `
};

const App = {
  components: {
    SampleOne
  },
  template: `
    <h1>Vue 3 Suspense</h1>
    <div class="app">
      <h3>Async/Promise with Success</h3>
      <Suspense>
        <template #default>
          <SampleOne />
        </template>
        <template #fallback>
          <div>Loading ...</div>
        </template>
      </Suspense>
    </div>
  `
};

createApp(App).mount("#app");

// Utils

async function fetchData() {
  return new Promise(async (resolve) => {
    const res = await fetch(
      `
       https://disease.sh/v3/covid-19/countries
       /?yesterday=true&strict=true
      `
    );
    setTimeout(async () =>
      resolve(await res.json()), 2000
    );
  });
}
```



COVID-19 Dashboard

Data by Country

Afghanistan
Cases: 37953
Recovered: 28016
Active: 8552

Albania
Cases: 8275
Recovered: 4184
Active: 3846

Algeria
Cases: 41068
Recovered: 28874
Active: 10770

Andorra
Cases: 1045
Recovered: 875
Active: 117

Angola
Cases: 2134
Recovered: 814
Active: 1226

Anguilla
Cases: 3
Recovered: 3
Active: 0

Antigua and Barbuda
Cases: 94
Recovered: 89
Active: 2

Argentina
Cases: 336802
Recovered: 245781
Active: 84173

Armenia
Cases: 42616
Recovered: 35907
Active: 5859

Aruba
Cases: 1534
Recovered: 446
Active: 1081

Australia
Cases: 24602
Recovered: 18759
Active: 5358

Austria
Cases: 25062
Recovered: 21406
Active: 2924

Figure 2: Async/Promise Unhandled Failure.

The wrapped component wasn't able to resolve the async/promise operation, and the component was never ready to be displayed on the screen. Therefore, the Suspense component keeps the fallback content showing. **Figure 2** shows the app running.

This example demonstrates the use of a Suspense component wrapping a component with an async/promise fetch operation failing without being handled.

Async/Promise Operation with Handled Failure

The third and last scenario is handling the async/promise request failure inside the wrapped component. The Suspense component gets the impression that the wrapped component has successfully completed its async/promise fetch operation and it's ready to be rendered. In this case, the Suspense component displays the wrapped component and hides the fallback content. **Listing 2** shows the complete source code used in this section.

Access this example on Suspense with Handled Failure here: https://codesandbox.io/s/suspense-handled-failure-yy181.

The async/promise fetch operation is now wrapped inside a try/catch block. The rejected Promise inside fetchData() function is handled and written into the console silently. Hence, the component is safe to be rendered and no exception or failure was propagated to the parent component. **Figure 3** shows the app running.

The Suspense component renders the component normally. However, the data that the REST API should return is not showing due to the failure.

This section was longer than anticipated. I wanted to show you the possible scenarios that the Suspense component can have while deciding to render content or fallback content on the page.

How Suspense Fits into Your App?

An architectural look into how the Suspense in Vue 3 fits in your app leads me to discuss the various options that are available while implementing it.

Suspense at the App Level

You can position the Suspense component inside the App.vue component so it wraps the entire app and controls its rendering process. **Figure 4** illustrates Suspense at the App level.

The Suspense component wraps the entire app. The app contains Views which, in turn, are composed of Components.

Suspense at the View Level

Instead of wrapping the entire app inside a Suspense component, you just wrap a single View in your app with it.

Therefore, you either wrap a single View or simply wrap the entire <router-view /> by a Suspense component. By doing so, it controls every single View in your app. **Figure 5** illustrates this.

The Suspense component wraps a View. This View is composed of a set of Components.

A View in a Vue app can be thought of as a Page that users can route to. A View is composed of a set of components that are usually not route-aware.

Suspense at the Component level

The final option is to let the Suspense component wrap one or more components inside a View. This option allows you to control the display of a single component at a time.

This option definitely gives you more control and allows your app to behave differently among the different Suspense components that it uses. This comes at the expense of having several Suspense components and is more complicated to manage and maintain. **Figure 6** illustrates this.

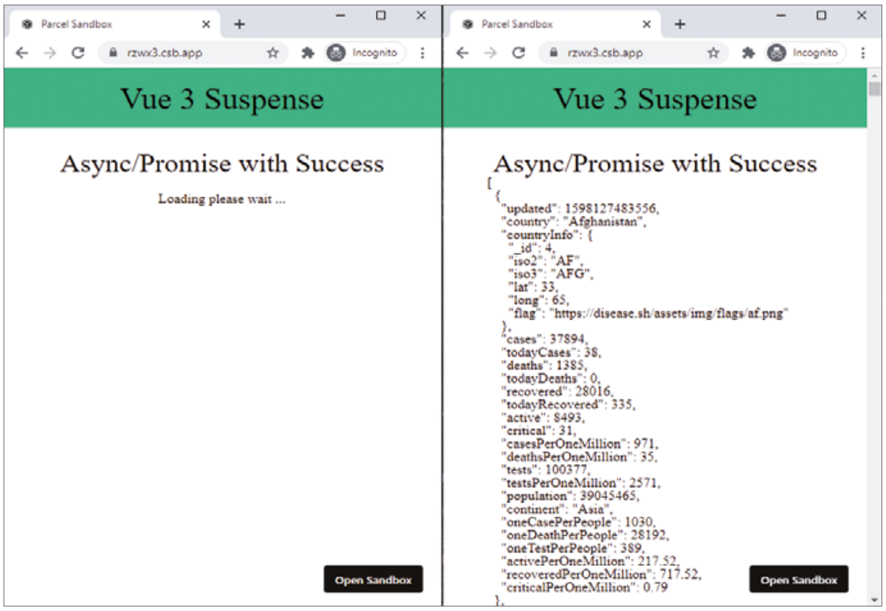
The Suspense component wraps all the Components inside a single View.

Demonstration

Now that you've had a full overview on how Suspense works in Vue 3 and how it's used in your app, let's switch gears and build something useful.

In this section, you'll build a basic COVID-19 Dashboard. The Dashboard displays COVID-19 data about each listed country. The information includes:

- Total cases
- Recovered cases
- Active cases



```
[
 {
  "updated": 1598127483556,
  "country": "Afghanistan",
  "countryInfo": {
   "_id": 4,
   "iso2": "AF",
   "iso3": "AFG",
   "lat": 33,
   "long": 65,
   "flag": "https://disease.sh/assets/img/flags/af.png"
  },
  "cases": 37894,
  "todayCases": 38,
  "deaths": 1385,
  "todayDeaths": 0,
  "recovered": 28016,
  "todayRecovered": 335,
  "active": 8493,
  "critical": 31,
  "casesPerOneMillion": 971,
  "deathsPerOneMillion": 35,
  "tests": 100377,
  "testsPerOneMillion": 2571,
  "population": 39045465,
  "continent": "Asia",
  "oneCasePerPeople": 1030,
  "oneDeathPerPeople": 28192,
  "oneTestPerPeople": 389,
  "activePerOneMillion": 217.52,
  "recoveredPerOneMillion": 717.52,
  "criticalPerOneMillion": 0.79
 },
```

Figure 3: Async/Promise with Handled Failure.

Listing 2: Async/Promise Operation with Handled Failure

```
import { createApp, ref } from "./vue.esm-browser";

const SampleThree = {
  name: "Sample Three",
  async setup() {
    const data = ref(null);

    try {
      data.value = await fetchData();
    } catch (e) {
      console.log(e);
    }

    return {
      data
    };
  },
  template: `
    <p style="text-align: center; padding-top: 20px;">
       This component is rendered without any data fetched!
    </p>
    <pre>{{ data }}</pre>
  `
};

const App = {
  components: {
    SampleThree
  },
  template: `
    <h1>Vue 3 Suspense</h1>
    <div class="app">
      <h3>Async/Promise with Success</h3>
      <Suspense>
        <template #default>
          <SampleThree />
        </template>
        <template #fallback>
          <div
           style="text-align: center; padding-top: 20px;">
            Loading please wait ...
          </div>
        </template>
      </Suspense>
    </div>
  `
};

createApp(App).mount("#app");

// Utils

async function fetchData() {
  return new Promise(async (resolve, reject) => {
    try {
      const res = await fetch(
        `
         https://diseases.sh/v3/covid-19/countries/
         ?yesterday=true&strict=true
        `
      );
      setTimeout(async () =>
         resolve(await res.json()), 2000
      );
    } catch {
      reject("Failed to load data");
    }
  });
}
```

I'm using the https://disease.sh/docs/ website to retrieve information that will populate my app. This website offers a modern REST API that can return JSON data to consume in your app. You can browse their documentation for more information on how to use their API.

Listing 3: Covid by Country Component

```
import { computed, ref } from "./vue.esm-browser";

export default {
  name: "CovidByCountry",
  props: {
    timeout: {
      type: Number,
      default: 3000
    }
  },
  async setup(props: any) {
    const data = ref(null);
    data.value = await fetchData(props.timeout);

    return {
      covid: computed(() => {
        return data.value;
      })
    };
  },
  template: `
   <div class="covid">
    <div class="covid-line"
         v-for="record in covid"
         :key="record.countryInfo._id">
      <span class="covid-country">
        {{ record.country }}
      </span>
      <ul>
        <li>Cases: {{ record.cases }}</li>
        <li>Recovered: {{ record.recovered }}</li>
        <li>Active: {{ record.active }}</li>
      </ul>
    </div>
    </div>
  `
};
```

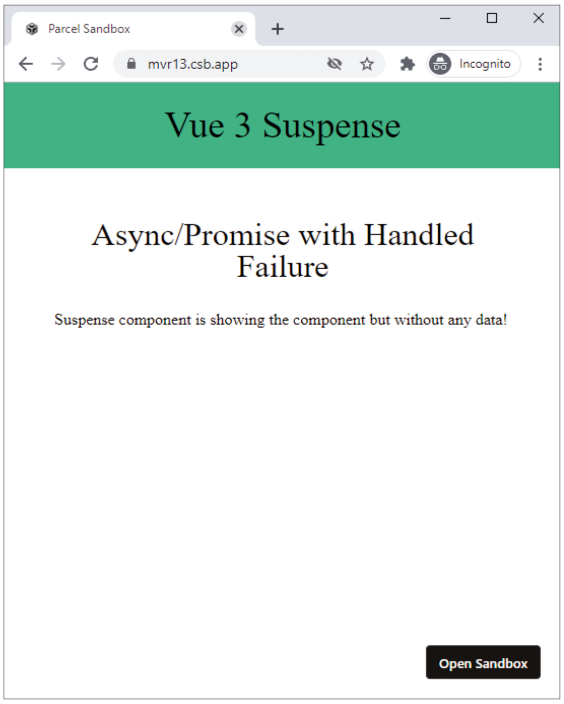
For this Dashboard, let's use the following HTTP GET request: https://disease.sh/v3/covid-19/countries/?yesterday=true&strict=true.

The request queries for all the countries having COVID-19. It retrieves the information from the day before.

To start with, let's build a function to fetch the data from the backend.

```
async function fetchData(
  timeout: number = 2000) {

  return new Promise(
   async (resolve, reject) => {
    try {
      const res = await fetch(`
       https://disease.sh/v3/covid-19/countries
     /?yesterday=true&strict=true`);

      setTimeout(async () => {
         resolve(await res.json());
       }, timeout);

    } catch (err) {
      reject(err);
    }
  });
}
```

The function uses the Fetch API (https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API) to send a request to the REST API. Before returning the results, the function simulates a delay by using the setTimeout() function with a timeout delay in milliseconds. The default value of the timeout is two seconds. In addition, a try/catch block wraps around the fetch() request call in order to handle any errors. This is essential to make sure your Promise can always resolve or reject.

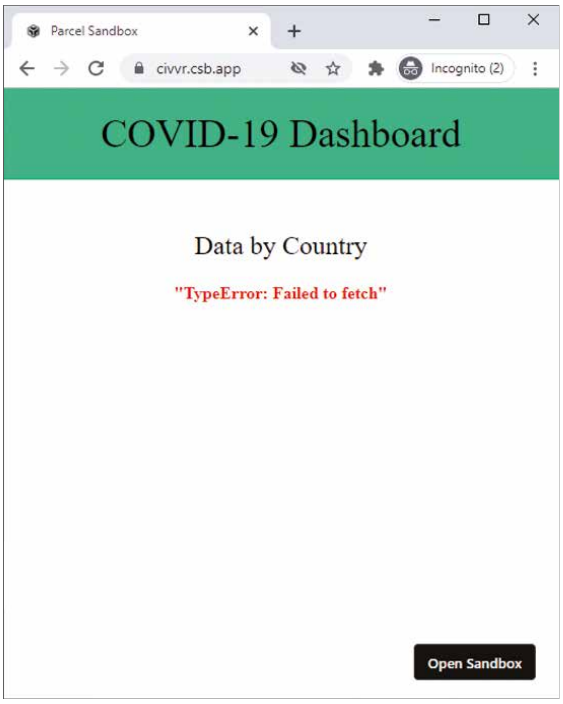


Vue 3 Suspense

Async/Promise with Handled Failure

Suspense component is showing the component but without any data!

Figure 4: Suspense at the App Level.



COVID-19 Dashboard

Data by Country

"TypeError: Failed to fetch"

Figure 5: Suspense at the View Level.

COVID by Country Component

Now let's build the component that displays the COVID-19 data on the page. **Listing 3** shows the complete code of this component.

The component accepts a single property to specify the timeout delay before displaying the results. It uses the Composition API by using the async setup() function.

Inside the function, it defines a ref variable named data. It then awaits the fetchData() function. Finally, it returns an object with a single computed() property named **covid** that holds a reference to the data coming back from the server.

The key point here is that the component is awaiting a promise inside an async setup() function. This is the main requirement for a component to work inside the Suspense component.

The component defines a basic template to display each country together with its results.

Add Suspense to the App Component

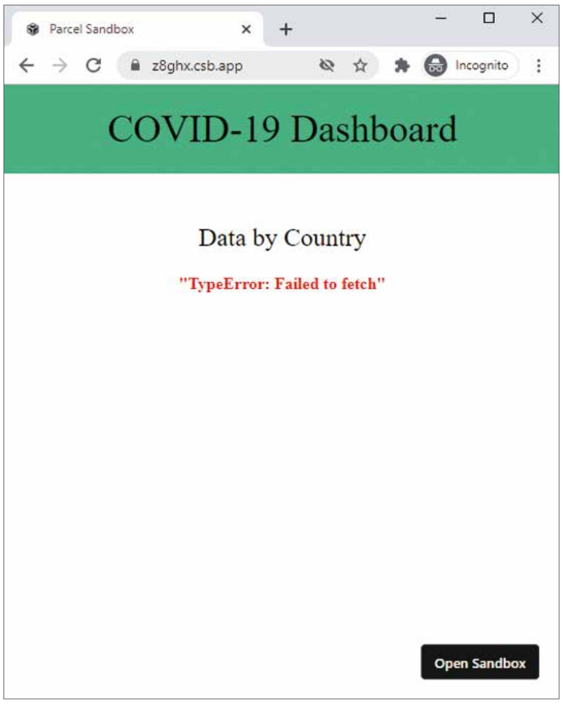
Let's switch to the App component and configure the Suspense component inside it. **Listing 4** shows the complete code for this component.

The component uses a Suspense component in its template. The Suspense component defines two templates: default and fallback.

The former hosts the CovidByCountry component that you defined above. The latter defines a message to let the user know that data is being retrieved from the back-end server and will be available soon.

The moment the data is retrieved and returned, the setup() function is resolved. The Suspense component hides the fallback template and displays the default template. The default template gets replaced by the CovidByCountry component. **Figure 7** shows the app running.

You can play with this example on the COVID-19 Dashboard: https://codesandbox.io/s/suspense-vue3-4vgle.



"TypeError: Failed to fetch"

Figure 6: Suspense at the component level.

Listing 4: Add Suspense to the App Component

```
import { createApp, ref } from "./vue.esm-browser";
import CovidByCountry from "./CovidByCountry";

const App = {
  components: {
    "covid-by-country": CovidByCountry
  },
  setup() {
    const timeout = ref(4000);

    return {
      timeout
    };
  },
  template: `
    <h1>COVID-19 Dashboard</h1>
    <div class="app">
      <h3>Data by Country</h3>
      <Suspense>
        <template #default>
          <covid-by-country :timeout="timeout" />
        </template>
        <template #fallback>
          <div style="padding-top: 20px;">
            Loading please wait ...
          </div>
        </template>
      </Suspense>
    </div>
  `
};

createApp(App).mount("#app");
```

Error Handling in Suspense

In this section, I'll look at how to handle errors that are thrown while the async setup() function waits for the fetchData() function to execute.

You have two options for handling errors when you're using the Suspense component.

- At the parent component level where the Suspense component lives
- At the child component that the Suspense component wraps

I'll look at both options to give you a solid understanding of how to handle errors with the Suspense component.

There are several ways to simulate a failure in an HTTP request. I'll choose the easiest for this demonstration and mess around with the Host name of the REST API request. The change affects the fetchData() function at this line:

```
const res = await fetch(`
   https://diseases.sh/v3/covid-19/countries
   /?yesterday=true&strict=true
`);
```

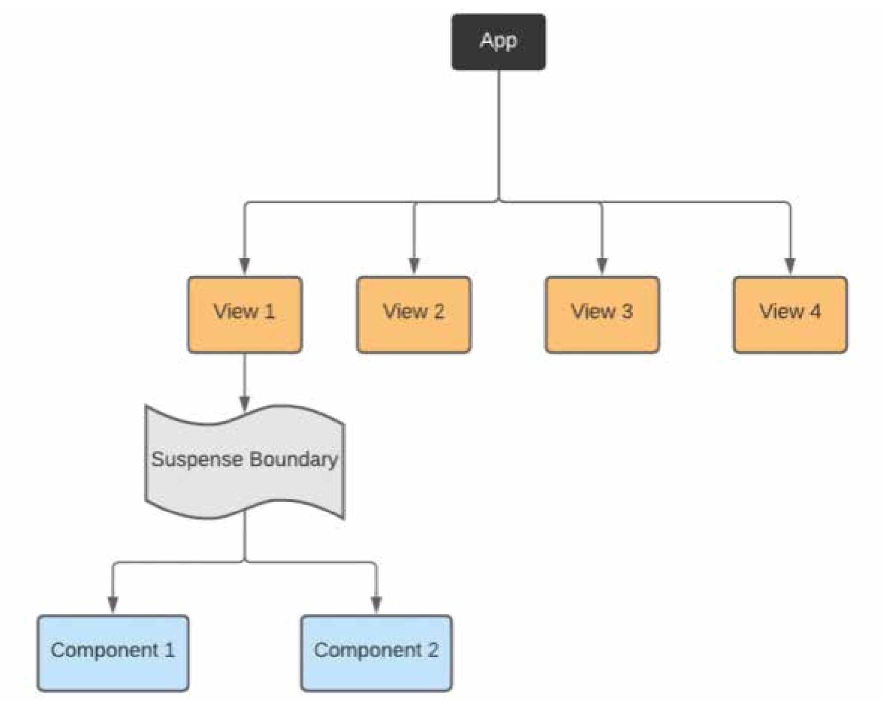


Figure 7: Preloading and loading of COVID-19 data.

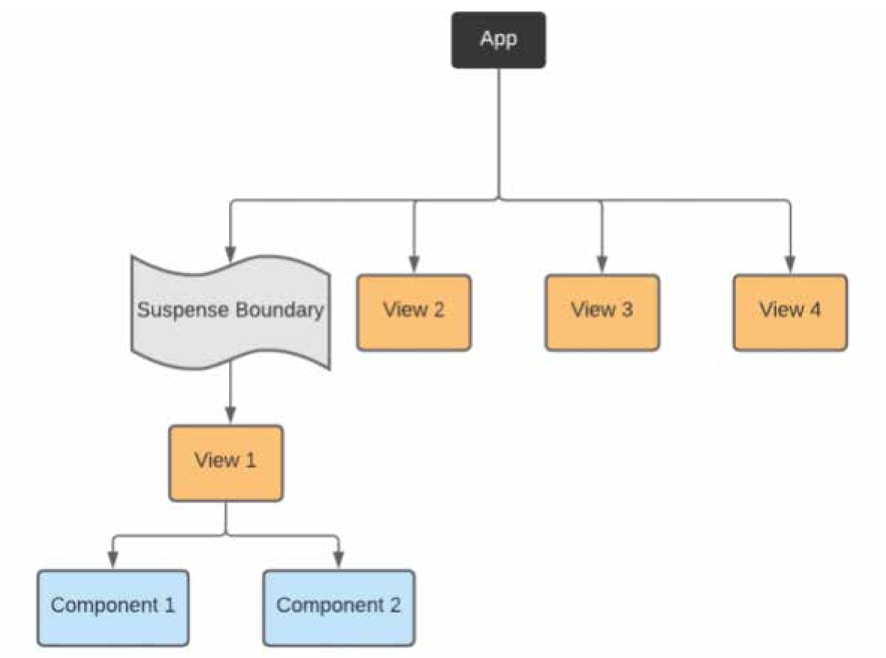


Figure 8: Handle errors at the parent component.

The request now uses **diseases.sh** instead of **disease.sh.**

The line above fails and throws an exception. The try/catch block handles the exception and **reject**s the Promise.

Option 1: Handle Errors at the Parent Component

Before I jump in and implement the solution, let me remind you of how the setup() function looks inside the CovidBy-Country component.

```
async setup(props: any) {
    const data = ref(null);
```

```
    data.value = await fetchData(props.timeout);

    return {
      covid: computed(() => {
        return data.value;
      })
    };
  },
```

The function awaits the fetchData() function without any try/catch block. If the fetchData() function rejects the Promise it returns, the setup() function fails, and the error bubbles up the Vue-Component-Tree.

Vue 3 offers the onErrorCaptured() lifecycle hook. Vue 2 introduced this hook back at version 2.5.0 of the framework. Vue 3 renames this hook and makes it available inside the Composition API.

Vue calls the onErrorCaptured() hook when an error from any descendent component is captured.

This hook perfectly fits the scenario here. Let's use it inside the App component (the component that hosts the Suspense component).

Let's modify the setup() function inside the App component to look like this:

```
setup() {
    ...
    const error = ref(null);

    onErrorCaptured(
      (e: Error) => (error.value = e)
    );

    return {
      error,
      ...
    };
  },
```

The onErrorCaptured() function receives three arguments: the error, the component instance that triggered the error, and a string containing information on where the error was captured. The hook can return false to stop the error from propagating further.

The hook assigns the error to a local ref variable named error.

Let's switch to the template and modify it a little in order to display the errors.

```
template: `
    <h1>COVID-19 Dashboard</h1>
    <div class="app">
      <h3>Data by Country</h3>
      <div class="space-up error" v-if="error">
        {{ error }}
      </div>
      <Suspense v-else>
        ...
      </Suspense>
    </div>
  `
```

When there's an error, the component hides the Suspense component and instead shows the error message. Otherwise, it shows the Suspense component when there is no error message.

Figure 8 shows the app running.

That's it!

Toy with this example on Handling Errors at the Parent Component here: https://codesandbox.io/s/suspense-handle-errors-vue3-z8ghx.

Now let's see how you can handle errors at the child component without touching the parent component.

Option 2: Handle Errors at the Child Component

To implement this option is much simpler. All you need to do is wrap the call to the fetchData() function with a try/catch block. Inside the catch(), assign the error to a local ref variable named **errors**.

```
async setup(props: any) {
    ...
    const error = ref(null);

    try {
      data.value =
         await fetchData(props.timeout);
    } catch (err) {
      error.value = err;
    }

    return {
      ...
      error
    };
  },
```

When you safely handle any errors at the child component level, the async setup() function resolves successfully. Hence, the Suspense component that hosts this child component displays the default template—the CovidByCountry component—and hides the fallback content. You are, more or less, tricking the Suspense component into believing all is well. In fact, when you handle the HTTP request calls properly, things are fine even though the data cannot be retrieved!

Let's have a look at the template now:

```
<div class="space-up error center" v-if="error">
   {{ error }}
</div>
<div v-else class="covid">
   <div class="covid-line"
        v-for="record in covid"
        :key="record.countryInfo._id">
     ...
   </div>
</div>
```

The error message is shown when there's an error. Otherwise, the template displays the list of all countries with COVID-19 data.

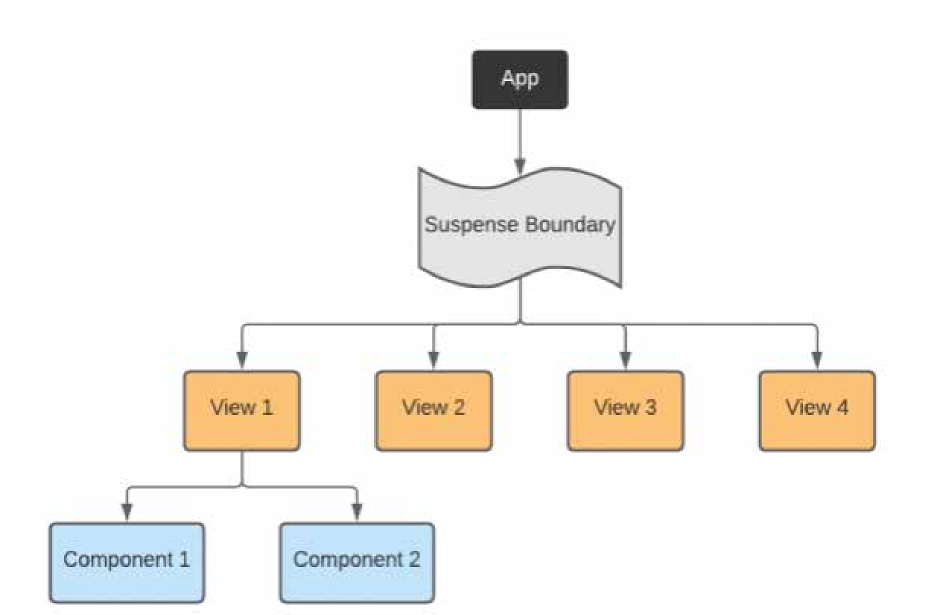


Figure 9: Handle errors at the child component.

Figure 9 shows the app running.

You can play with this example on Handling Errors at the Child Component here: https://codesandbox.io/s/suspense-handle-errors-child-vue3-civvr.

Bonus: Custom Suspense Component with Error Handling

I've included this section to clean up the way I handle errors with the new Suspense component.

By default, the Suspense function can either display the default template or the fallback template. It leaves out handling errors and lets the developer do it their own way.

In this section, I'll propose a rather opinionated approach on how to extend the Suspense component to offer a third template: the error template.

My approach can be summarized in three important points:

- Wrap the Suspense component inside a parent component.
- Handle the onErrorCaptured() inside the parent component.
- Use Vue Slots to render the default, fallback, and error templates.

To learn more about the Vue Slots, check out my article: Content Distribution in Vue JS (https://labs.thisdot.co/blog/content-distribution-in-vue-js).

Wrap the Suspense Component

Let's wrap the Suspense component inside a new component named SuspenseWithErrors.

```
export default {
  name: "SuspenseWithErrors",
  template: `
    <Suspense>
```

Vue 3 Docs

The Vue team provides and maintains a detailed documentation on the framework that you may always use as a reference: https://v3.vuejs.org/guide/introduction.html

Listing 5: Using the SuspenseWithErrors Component

```
template: `
...
<SuspenseWithErrors>
  <template #error="props">
    <p class="center error space-up">
      {{ props.error }}
    </p>
  </template>
  <template #default>
    <covid-by-country :timeout="timeout" />
  </template>
  <template #fallback>
    <div class="space-up">
      Loading please wait ...
    </div>
  </template>
  </Suspense>
</div>
`
```

COVID-19 API

If you're interested in learning more about the COVID-19 API used in this article, feel free to check it at [Disease.sh] (https://disease.sh/).

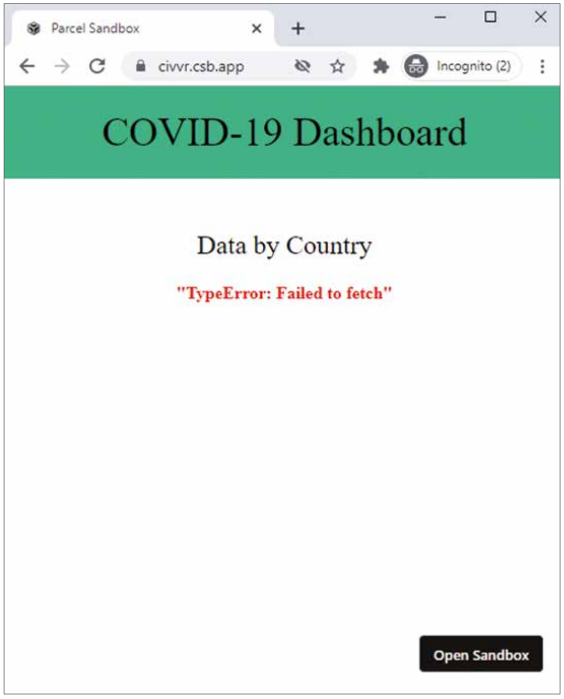


Figure 10: SuspenseWithErrors in action.

```
      <template #default>
      </template>
      <template #fallback>
      </template>
    </Suspense>
  `
};
```

Handle the onErrorCaptured() Hook

Now let's handle the onErrorCaptured() hook inside this new component.

```
setup() {
    const error = ref(null);

    onErrorCaptured(
       (err: Error) => (error.value = err)
    );
```

```
    return {
      error
    };
  },
```

Add the Vue Slots

Finally, let's use the Vue Slots to cater for the three templates:

```
<slot
   name="error"
   :error="error"
   v-if="error"
/>
<Suspense v-else>
   <template #default>
      <slot name="default" />
   </template>
   <template #fallback>
      <slot name="fallback" />
   </template>
</Suspense>
```

If there's an error, the Slot named **error** is displayed. Otherwise, the Suspense component is displayed. I'm introducing two new slots for the default and fallback content. This approach makes the component more flexible.

Note how I'm binding an error property on the error slot to the actual value of the error. Vue calls this Scoped Slots. This way, the component consuming the SuspenseWithErrors component can directly access the error message.

Let's see this component in action!

Using the SuspenseWithErrors Component

First of all, make sure the CovidByCountry component isn't handling any errors as follows:

```
async setup(props: any) {
  ...
  data.value =
     await fetchData(props.timeout);
  ...
},
```

Second, make sure the App component that's hosting the Suspense component isn't handling any errors.

At the App component level, import the new component named SuspenseWithErrors.

```
...
import
   SuspenseWithErrors
from
   "./SuspenseWithErrors";

const App = {
  components: {
    ...
    SuspenseWithErrors
  },
```

Now let's change the template of the App component to use this new one. **Listing 5** shows the complete code of this template.

Notice how the SuspenseWithErrors component now supports three templates: error, default, and fallback.

In addition, note how the Error template accesses its Error property, that the SuspenseWithErrors component binds on it by binding the name of the slot to the Props property. The Props object holds any property bound to this slot. You access the Error property by using props.error.

Figure 10 shows the app running.

You can play with this example on Suspense With Errors here: https://codesandbox.io/s/suspense-with-errors-wrapper-bijok.

Conclusion

Let's be honest here, for every Vue app I've developed, I've had to always include a Vue component with boilerplate logic to show or hide the main component once its data is loaded and ready for display on the page.

In this case, Vue 3 Suspense eliminates it. The API is pretty good, straightforward and easy to use.

Still, I believe this new feature comes with its own "side-effects." The major one is deciding where in your app to use the <Suspense /> component. In other words, deciding on the level of isolation is the real challenge.

In future articles, I look forward to discussing the challenges that arise and sharing solutions with you as I come across them. Until then, happy coding.

Bilal Haidar

(Continued from 74)

person for a job. Winston Churchill was a prime example of that. The slippery slope is when the leader lets their duty to others become subordinated to their notion of self-importance. Generally, a good leader is aware.

Aware: Knowledge or Perception of a Situation or Fact

Leaders know the facts and they act on the facts. Perhaps more importantly, they communicate the facts. In furtherance of their duty, a leader sees to it that those facts are communicated to the people on whose behalf they act.

Empathy: The Ability to Understand and Share the Experiences and Feelings of Another

There's a saying that hypocrisy has a way of revealing itself when the shoe is on the other foot. Leaders take the time to understand other points of view. Empathy doesn't mean there must be agreement. Understanding is about having a sense of why others think and feel as they do.

Assessing Leadership

Is the person being evaluated a leader? Don't confuse leadership with their official position. Often, those two things coincide, and all too often, they don't. Leadership can be objectively assessed. It's important to note that words like "perfection" don't apply. Nobody is perfect. Nobody is correct on 100% of things 100% of the time.

Another word that I don't think applies is "inspire." What makes somebody a leader is what they do on matters they have control over. Inspiration is about how others feel. Leaders don't have control over how others feel or how they're motivated. Leadership enables—but that's a different concept. How you feel and how you're motivated, that's up to you. That's all about leading yourself. In this regard, there are definite boundaries between the objective and the subjective evaluation of leadership. Make no mistake that on an objective basis, without getting into specific idiosyncratic behavior and personalities, you can objectively ascertain whether the leadership label applies by squaring observable fact to defined terms. A person's actions either comply with the definition or it doesn't. There is no in-between.

John V. Petersen

CODE COMPILERS

CODE

Nov/Dec 2020
Volume 21 Issue 6

Group Publisher
Markus Egger

Associate Publisher
Rick Strahl

Editor-in-Chief
Rod Paddock

Managing Editor
Ellen Whitney

Content Editor
Melanie Spiller

Editorial Contributors
Otto Dobretsberger
Jim Duffy
Jeff Etter
Mike Yeager

Writers In This Issue
Bilal Haidar
Sahil Malik
Rick Strahl
Shawn Wildermuth
Wei-Meng Lee
Paul Sheriff
John V. Petersen

Technical Reviewers
Markus Egger
Rod Paddock

Production
Franz Wimmer
King Laurin GmbH
39057 St. Michael/Eppan, Italy

Printing
Fry Communications, Inc.
800 West Church Rd.
Mechanicsburg, PA 17055

Advertising Sales
Tammy Ferguson
832-717-4445 ext 26
tammy@codemag.com

Circulation & Distribution
General Circulation: EPS Software Corp.
Newsstand: The NEWS Group (TNG)
Media Solutions

Subscriptions
Subscription Manager
Colleen Cade
ccade@codemag.com

US subscriptions are US $29.99 for one year. Subscriptions outside the US are US $49.99. Payments should be made in US dollars drawn on a US bank. American Express, MasterCard, Visa, and Discover credit cards accepted. Bill me option is available only for US subscriptions. Back issues are available. For subscription information, e-mail subscriptions@codemag.com.

Subscribe online at
www.codemag.com

CODE Developer Magazine
6605 Cypresswood Drive, Ste 425, Spring, Texas 77379
Phone: 832-717-4445

What Makes a Leader: An Objective Analysis

As a lawyer and a software engineer, I endeavor to base assessments on facts and empirical evidence. In times of crisis, leadership at all levels is scrutinized. Can leadership be objectively assessed? Or is leadership something that can only be subjectively assessed? That question turns on whether there are definitions that exist separate and distinct from the facts. If such definitions exist, from there, we can apply those definitions to the facts to formulate a conclusion.

The definitions offered herein are not subject to debate as they're straight out of the dictionary. As for whether the qualities I'm outlining here are hallmarks of leadership, that might require subjective evaluation. Then again, if we look at recognized leaders in history, what qualities do they tend to possess? If someone doesn't possess the qualities discussed herein, are they more or less likely to be truly regarded as leaders?

What are the qualities of leadership?

Defining the Words

A good place to start is with the word "lead" itself.

Lead: To be a route or means of access to a particular place or a in a particular direction. To be in charge of or in command of.

The concept of leadership is about one person acting on behalf of one or more other people. Leadership is also about going in some direction to achieve some objective. That implies that leadership is about moving in what could be judged to be the correct direction so that the correct objective can be achieved.

One approach in describing what something is can be to provide an example of what it's not.

Mislead: To cause (someone) to have a wrong idea or impression about someone or something.

Two additional words that come to mind are Accountable and Responsible:

- **Accountable:** Required or expected to justify actions or decision; responsible
- **Responsible:** Having an obligation to do something, or having control over or care for someone, as part of one's job or role

The implication of being accountable and responsible is that there is some duty.

Duty: A Moral or Legal Obligation; Responsibility

Because the context is about acting on the behalf of others, going in a chosen direction to achieve an objective, a leader must show some regard for the future. When acting on behalf of others, duty is all about the best interest of others. Leaders avoid conflict of interest from subordinating other interests to their own self-interests, with respect to the future.

Prudent: Acting with or Showing Care and Thought for the Future

Duty and prudence imply that a leader is a fiduciary, which is a relationship that has **trust** as its cornerstone. When a leader advocates a point of view, we know instinctively whether they believe what they say. It also implies how decisions get made.

Equitable: Fair and Impartial

Fair: Decisions are made based on facts, and always in the best interest of whomever is to be served regardless of whether it is in the best interest of the person tasked with making the decision. That, however, is not enough. **Impartial**: Such decision must be unbiased and not in favor of a specific individual or group of individuals.

Communication is always key. To act on behalf of others requires sound communication.

Respond: Act or Behave in Reaction to Someone or Something

How do good leaders respond? Like leading vs. misleading, it's good to flip the problem on its ear and review from the opposite perspective.

React: Responding with hostility. If a leader is to engender trust such that the people for whom he or she acts on behalf can be guided in a correct direction, there can't be an adversarial relationship between the leader and the people being led. Leaders respond; they don't react.

So far, nothing speaks to knowledge and skill. But there's a fundamental threshold issue to address, it's whether a person is capable of being a leader. This is over and above the technical qualities required for the task at hand.

Competence: Having the Necessary Ability, Knowledge, or Skill to Do Something Successfully

Somebody can be physically at the top of some organizational structure by virtue of some other process. Do they occupy the position held solely because of merit? Or are politics involved? The person may be expert at the tasks accomplished by the team, and yet totally incompetent insofar as being a leader is concerned. In management, this is the Peter Principle where people tend to rise to their level of incompetence. There's also the notion that past success isn't a guarantee of future success.

Of course, no one person is an expert in all things. A leader defers to others who know more about an issue.

Deference: Humble Submission and Respect

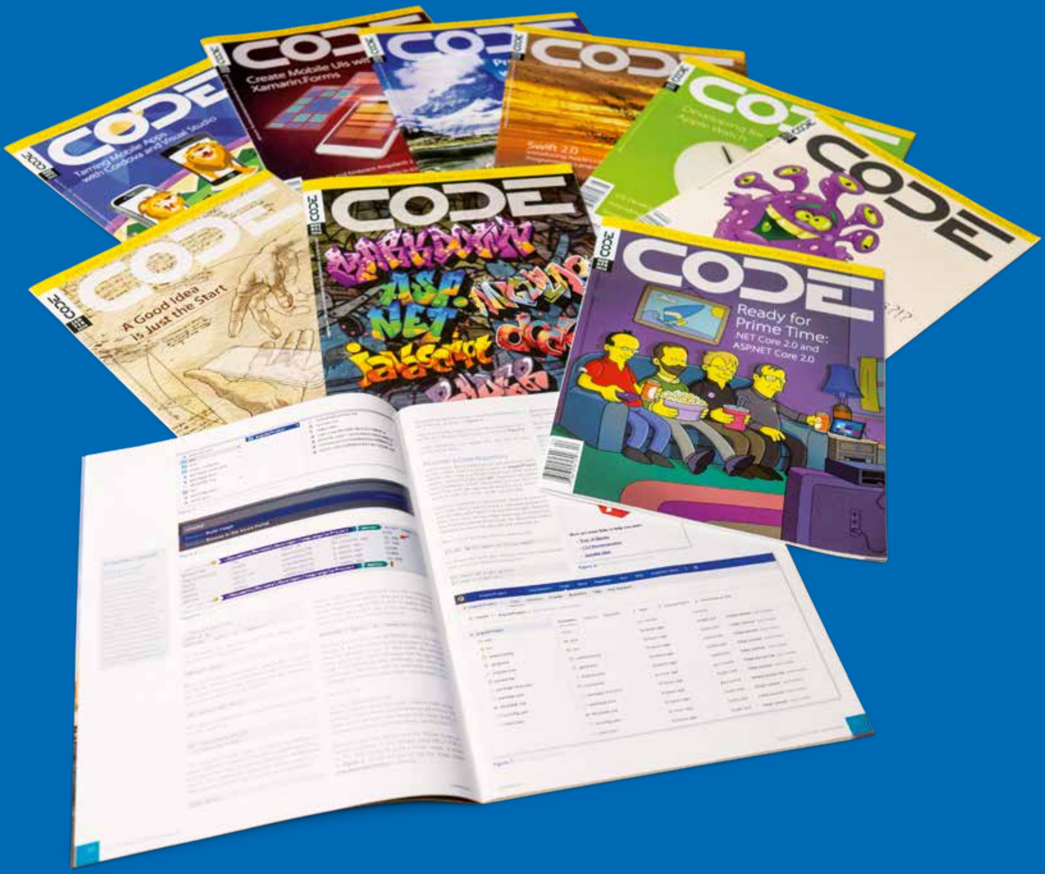
Deference is about knowing what one doesn't know and acknowledging that fact so that the person who's responsible for achieving some objective can fulfill that duty. Often, leading means stepping back to let others take the reins on a particular issue. A hallmark of recognized great leaders is that they don't ever seek to portray the myth that they are the smartest person in the room on all topics.

This is often referred to as leading from behind and servant leadership. A leader understands that their role can often be one of support and to do the blocking and tackling in order for others to be able to do their jobs. This is where ego becomes an issue. Ego is about a person's self-esteem or self-importance.

All leaders have egos. They believe they can do the job and they often believe they are the best

(Continued on page 73)

CODE
MAGAZINE



KNOWLEDGE
IS POWER!

Sign up today for a free trial subscription at **www.codemag.com/subscribe/DNC3SPECIAL**

codemag.com/magazine
832-717-4445 ext. 8 • info@codemag.com

HERE ARE
750 BILLION REASONS
WHY DEVS SHOULD
READ THIS AD.

Presenting Fortellis, the open-exchange marketplace serving the $750-billion auto industry.

On the Fortellis platform, APIs can be securely published, conveniently administered and consumed at scale by a continually expanding universe of developers, dealers and manufacturers. You and your APIs and apps will be connected to an industry with a voracious appetite for them, and we'll even handle the integrations and go-to-market strategy.

Aren't you glad you kept reading? Read more at **Fortellis.io.**

OUR PLATFORM:

40+
Active APIs

2K+
Active Users

10M+
API Transactions

FORTELLIS
Automotive Commerce Exchange™

2020 Fortellis is a registered trademark of CDK Global, LLC. 20-1526