Intro to R, Bots, Data Warehousing, Node Streams, Azure Skyline

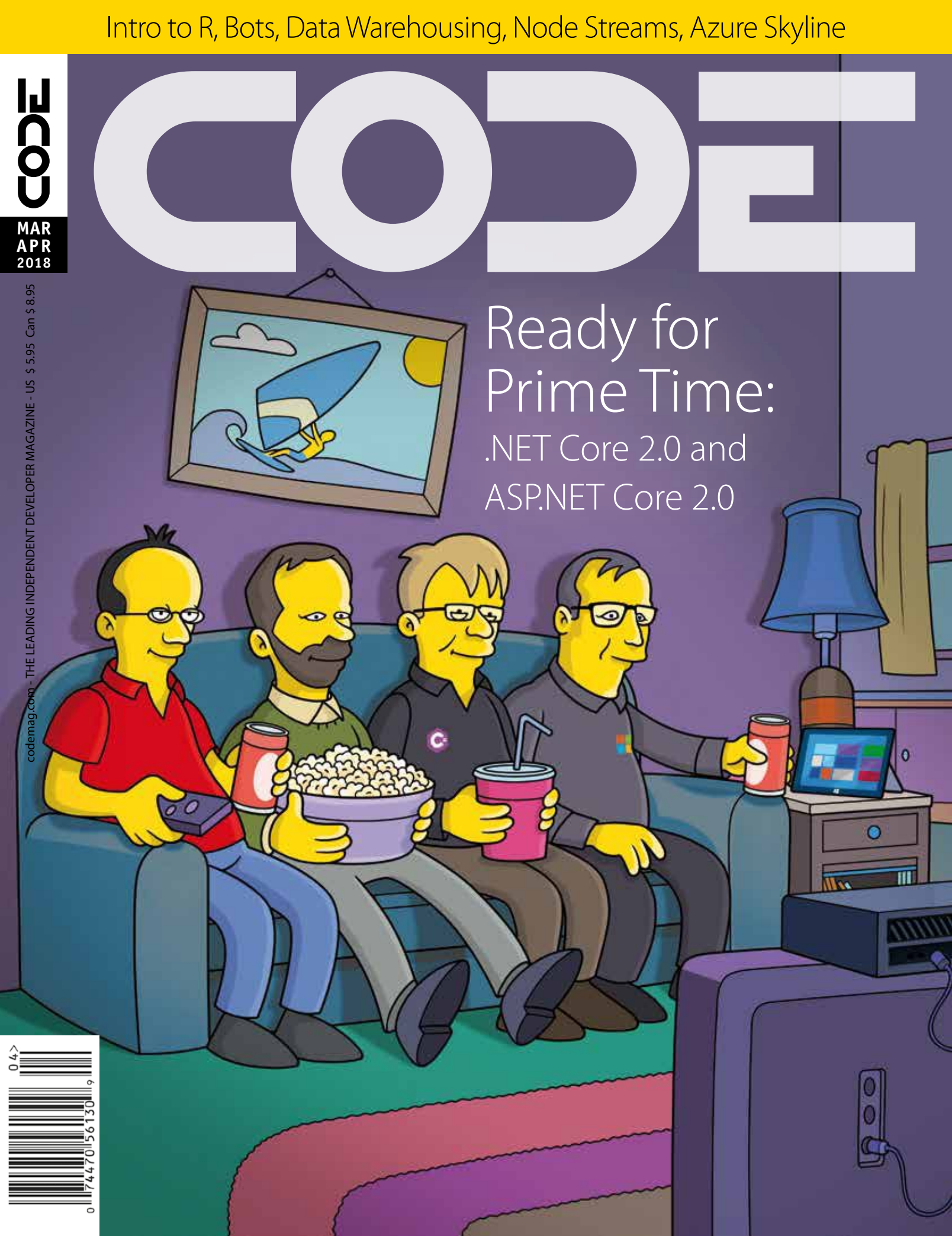
CODE

MAR
APR
2018

codemag.com - THE LEADING INDEPENDENT DEVELOPER MAGAZINE - US $ 5.95 Can $ 8.95

Ready for Prime Time:

.NET Core 2.0 and ASP.NET Core 2.0



04
0 74470 56130 9

Features

Columns

Departments

About the Cover:

.NET Core 2.0 and ASP.NET Core 2.0 are ready for prime time, and what better way to celebrate than to put some of the Microsoft people responsible for it on the world's favorite animated couch? Earn a free subscription for yourself and/or your friends if you know who they are!
Hint: That's Scott Guthrie on the left.
Go to www.codemag.com/subscribe/guthrie-lastname-lastname-lastname and fill in the last names of the others on the couch.

Don't know who they are?
Check out our Facebook page (www.facebook.com/CODEMagazine) and our Twitter account (@CODEMagazine) for hints.

US subscriptions are US $29.99 for one year. Subscriptions outside the US pay US $44.99. 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.

Subscribe online at codemag.com

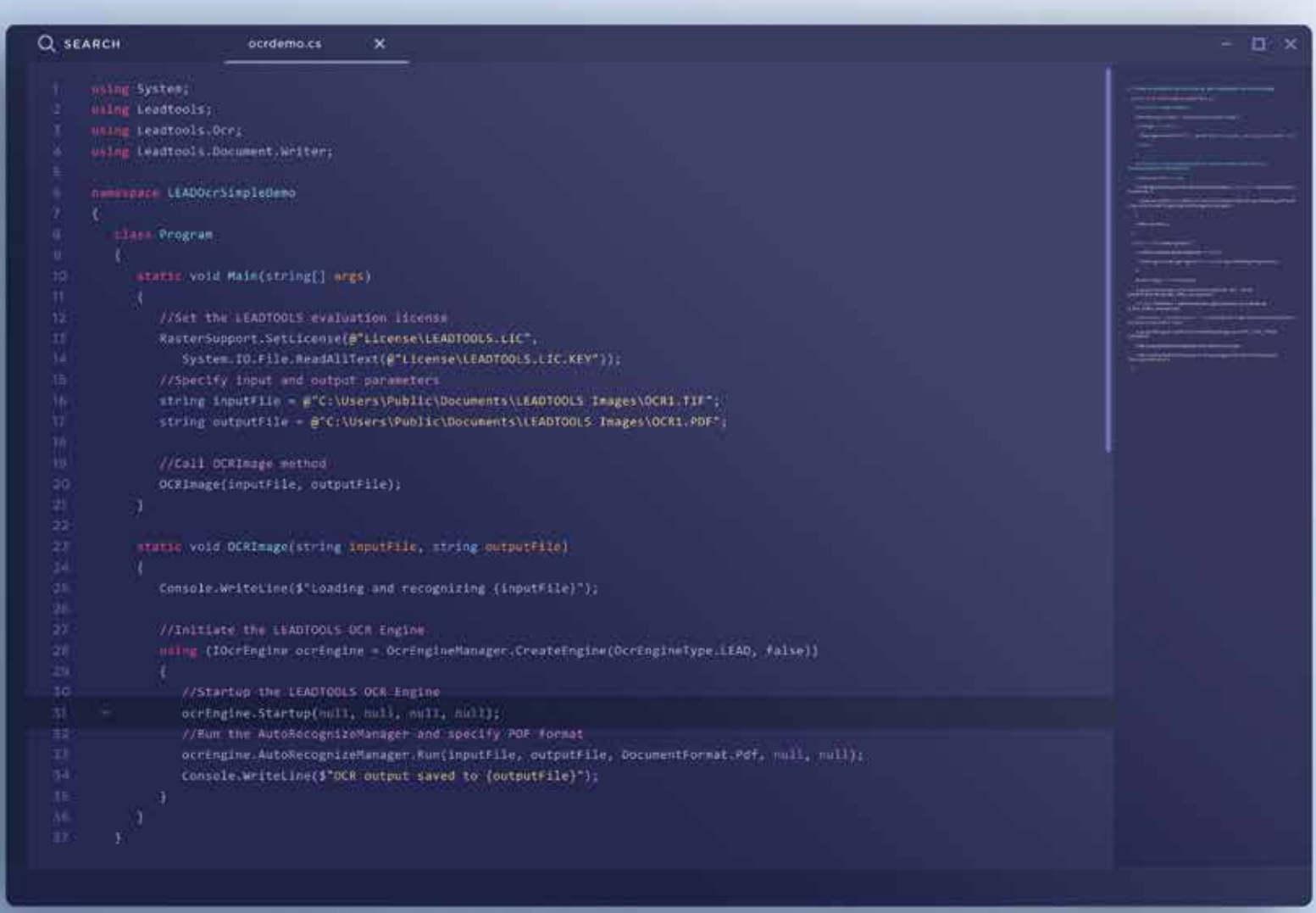
CODE Component Developer Magazine (ISSN # 1547-5166) is published bimonthly by EPS Software Corporation, 6605 Cypresswood Drive, Suite 300, Spring, TX 77379 U.S.A. POSTMASTER: Send address changes to CODE Component Developer Magazine, 6605 Cypresswood Drive, Suite 300, Spring, TX 77379 U.S.A.

Canadian Subscriptions: Canada Post Agreement Number 7178957. Send change address information and blocks of undeliverable copies to IBC, 7485 Bath Road, Mississauga, ON L4T 4C1, Canada.

LEADTOOLS V20

WEB MOBILE DESKTOP SERVER

WHAT DOES YOUR CODE LOOK LIKE?



Above is an entire OCR application using LEADTOOLS

That's it. Clean and simple code that produces a fast and accurate OCR app backed by the most powerful OCR SDK. Download our free evaluation and see how easy it is to use LEADTOOLS for your document, medical, and multimedia projects.

Microsoft .NET · Xamarin · Windows · Java · Linux · iOS · macOS · Android · HTML5

DOWNLOAD OUR FREE EVALUATION
WWW.LEADTOOLS.COM

LEAD TECHNOLOGIES INCORPORATED

SALES@LEADTOOLS.COM
800.637.1840

Ending Malaise

A few weeks ago, I had an existential crisis. For some reason, I just wasn't having fun writing code. To put this in proper context, I absolutely LOVE writing code and this caused me to question everything. I've had this feeling before but not for a prolonged amount of time. This time it was different and

I wasn't sure how I'd get myself to snap out of it. Both 2016 and 2017 were very busy years for my development team as we took a lot of blood, sweat, and tears to deliver two major projects for a client. Accompanying these two major projects was a normal dose of development work for other clients, mentoring, and trying to keep a semblance of a work/life balance. It's my belief that this contributed to my internal crisis. The interesting thing was how long the crisis went on and how quickly it disappeared.

Flash back to a few weeks ago. We were working on another set of major projects for our clients. One project is an API for one of the applications we delivered last year. This API is accessed from a new pair of mobile applications constructed for the Android and IOS operating systems. When building APIs, I like to spend time consuming those APIs from actual application code versus merely testing them in unit-testing frameworks or API development tools like Postman. It was during this development process that my breakthrough occurred.

I decided that I'd take time to investigate how our API would be consumed in Swift (the language for the IOS app). I updated my version of XCODE to the latest and did what all good developers do when learning a new technology. I Googled it. I searched for content on consuming REST services using the Swift programming language. I found many interesting posts that met my needs and set forth to communicate with my API. This process was very frustrating and yet I LOVED every minute of it. It was at this point that I realized why I was in such a malaise.

I hadn't had spent time doing what I really LOVE and the one thing that's essential to my career: learning new things. The opportunity to learn new things is what gets me out of bed in the morning and is probably the most fulfilling aspect of my job. It's funny how a frustrating development session can turn into something so rewarding.

Many years ago, I wrote an editorial about "Sharpening Your Ax." In that editorial, I talked about taking time away from working to refresh the batteries. Sometimes refreshing the batteries doesn't mean taking time away from work but adding new tools/ideas to your knowledge set.

Figure 1: Slump enders

This issue of CODE Magazine has some new directions from some of our more seasoned authors. Paul Sheriff writes about HTML grids instead of Angular, Sahil Malik writes about BOTS instead of SharePoint, and Wei-Meng Lee writes about the R programming language. Each of these authors either consciously or sub-consciously decided to extend their writing in new directions. What a great way to prevent fatigue and malaise.

I ran an early version of this editorial by my rockstar editor Melanie to see what she thought, and she pointed out that some people use the arts as a way to get out of a slump. This was a big DUH moment for me. **Figure 1** shows two recent purchases I made just before I made my breakthrough. It looks like I already had the potential solution to my problem but didn't even know it. To further expound on what Melanie said: This is part of the process of life-long learning. I love that concept. We as developers have chosen a path that requires us to be life-long learners and part of this process is learning new things. That's pretty Zen if you ask me.

You might be asking yourself at this point if my malaise/crisis returned. The answer is no. After recognizing the source of the malaise, I continued to challenge myself with new tools. I spent time doing nerdy stuff like manipulating SOAP headers with a SoapExtension class, did a little more work learning Swift, and looked into Python as another programming tool to add to my repertoire.

Rod Paddock
CODE

Data Quality Made Easy.
Your Data, Your Way.

Melissa provides the full spectrum of data quality to ensure you have data you can trust.

We profile, standardize, verify, match and enrich global **People Data** – name, address, email, phone, and more.

Our **data quality** solutions are available on-premises and in the Cloud – fast, easy to use, and powerful developer tools, integrations and plugins for the **Microsoft** and **Oracle Product Ecosystems.**

Microsoft SQL Server Integration Services · Microsoft Dynamics CRM · Microsoft .NET · ORACLE Forms · Java · ORACLE PL/SQL

Start Your Free Trial

www.melissa.com/code

Melissa Data is now Melissa.
See What's New at www.Melissa.com

1-800-MELISSA

melissa
GLOBAL INTELLIGENCE

Eliminate HTML Tables for Better Mobile Web Apps

If you're a Web developer, you know you should target your Web applications to be "mobile first." This means that your Web applications should look great on a mobile device, and good, if not great, on a desktop browser. There are many techniques you can use to help you develop for mobile. One technique is to use Bootstrap to give yourself responsive styles that change based on the device being used. Another technique, and the focus of this article, is to eliminate HTML tables.

Paul D. Sheriff
www.fairwaytech.com

Paul D. Sheriff is a Business Solutions Architect with Fairway Technologies, Inc. Fairway Technologies is a premier provider of expert technology consulting and software development services, helping leading firms convert requirements into top-quality results. Paul is also a Pluralsight author. Check out his videos at http://www.pluralsight.com/author/paul-sheriff.

HTML tables, when not used correctly, can cause problems with small screens such as those found on smart phones. This article shows you how to rework your HTML tables to fit better on mobile devices. In addition, you'll learn to use Bootstrap panel classes to completely eliminate HTML tables.

The Problem with HTML Tables

HTML tables are used by many Web developers because they're easy to program, and provide a way for users to see a lot of information like they would on a spreadsheet. But just because something is easy to use and conveys a lot of data, doesn't necessarily mean it's the best tool. There are many reasons why an HTML table isn't suitable for user consumption.

- A table presents too much data on the page, so the user has too much to concentrate upon.
- A user's eyes become fatigued after staring at rows and columns of data much more quickly than when data is spread out.
- It's hard for a user to distinguish between the data in each column because each column is uniform and nothing stands out.
- On a mobile device, the user frequently needs to pan right and left to see all the data. This leads to an annoyed user, and is very unproductive.

HTML Table on Desktop versus Mobile

In **Figure 1**, you see a list of tabular product data. This renders nicely on a normal desktop browser because the user has a lot of screen real-estate and they don't need to scroll left and right to see all the data.

Look at this same page rendered on a smart phone, as shown in **Figure 2**. The user is only able to see the left-most column of the table. If they don't know that they can scroll to the right, they're missing some important information. On some mobile browsers, the page may render the complete table, but it's so small that it's hard to read. Either way, the user is forced to interact with their phone to view the data. They must scroll left to right, or maybe pinch or spread with their fingers.

Create an MVC Project

If you wish to follow along creating the sample for this article, create a new MVC project using Visual Studio. Name the project **AlternativeTable**. Once you have a new MVC project, add three classes into the \Models folder. The names for each of these classes are **Product**, **ProductManager**, and **ProductViewModel**. Instead of using a database, create some mock data in the ProductManager class. The Product class is shown in the following code snippet:

```
public class Product
{
  public int ProductId { get; set; }
  public string ProductName { get; set; }
  public DateTime IntroductionDate { get; set; }
  public string Url { get; set; }
  public decimal Price { get; set; }
}
```

Several lines of the ProductManager class are shown in **Listing 1**. You need to add a few more Product objects into the list so you can display several rows of data. Or, see the sidebar for how to download the complete sample. You can then copy the ProductManager.cs class into the \Models folder to have several product objects to display while running this sample.

The last class is a view model that's called from the MVC Controller. Using an MVVM approach to development provides for a nice separation of concerns in your applications. It's also very easy to bind properties in your CSHTML pages to your view model classes. The ProductViewModel class is shown in the following code snippet.

```
public class ProductViewModel
{
  public List<Product> Products { get; set; } =
     new List<Product>();

  public void LoadProducts() {
    ProductManager mgr = new ProductManager();

    Products = mgr.Get();
  }
}
```

The MVC Controller

You need a MVC controller to load the data and feed that data to the CSHTML pages used to render your page of product data. Right mouse-click on the Controllers folder, select **Add** > **Controller... Select MVC 5 Controller – Empty** from the list of templates and click the Add button. Change the name to **ProductController** and click the OK button. Write the following code in the Index method.

```
public ActionResult Index()
{
  ProductViewModel vm = new ProductViewModel();

  vm.LoadProducts();

  return View(vm);
}
```

This method creates an instance of the ProductViewModel class and calls the LoadProducts method to build the Products property in the view model. The CSHTML page you are going to build uses the Products property of the View Model passed to it to build the HTML table.

The HTML Table View

Create a new folder under the \Views folder called \Product. Right mouse-click on this folder name and select **Add > MVC 5 View Page with Layout (Razor)** from the menu. Set the name to **Index** and click the OK button. When presented with a list of layouts, choose **_Layout.cshtml** from the dialog box and click the OK button. Write the code shown in **Listing 2**.

As you can see from **Listing 2**, there's nothing out of the ordinary for this table. You use Bootstrap table classes to help with styling the table. You loop through the collection of product data in the Products property of the ProductViewModel class. Each time through the loop, display the appropriate data from the Product class in each <td> of the table. You should be able to run the application at this point and see your table of product data appear. If you wish to see what this page looks like on a smart phone, you can purchase Electric Mobile Studio 2012 from Electric Plum at https://www.electricplum.com/studio.aspx.

Before you continue with this article, please copy the contents of your CSHTML page into notepad, or save it to another file. You're going to use this Razor code later in this article.

Remove Table-Responsive Class

There's one more thing to try before you move onto the next section. Remove the attribute *class="table-responsive"* from the surrounding the <table> element. Re-run the application and look at the result. You should now see more of the table, but the data is now wrapped within the columns. This can give you a little more data showing in the table, but you're right back to having too much data in too little space. If you have several more columns, it also forces the user to scroll left and right to see all of the data.

Reduce Table to Two Columns

If you reduce your table to two columns, you find that it fits on a small device screen just fine (See **Figure 3** and **Figure 4**). Make the changes to the page you just created by first modifying the columns in the <thead> area, as shown in the following code snippet:

```
<thead>
  <tr>
    <th>Product Information</th>
```

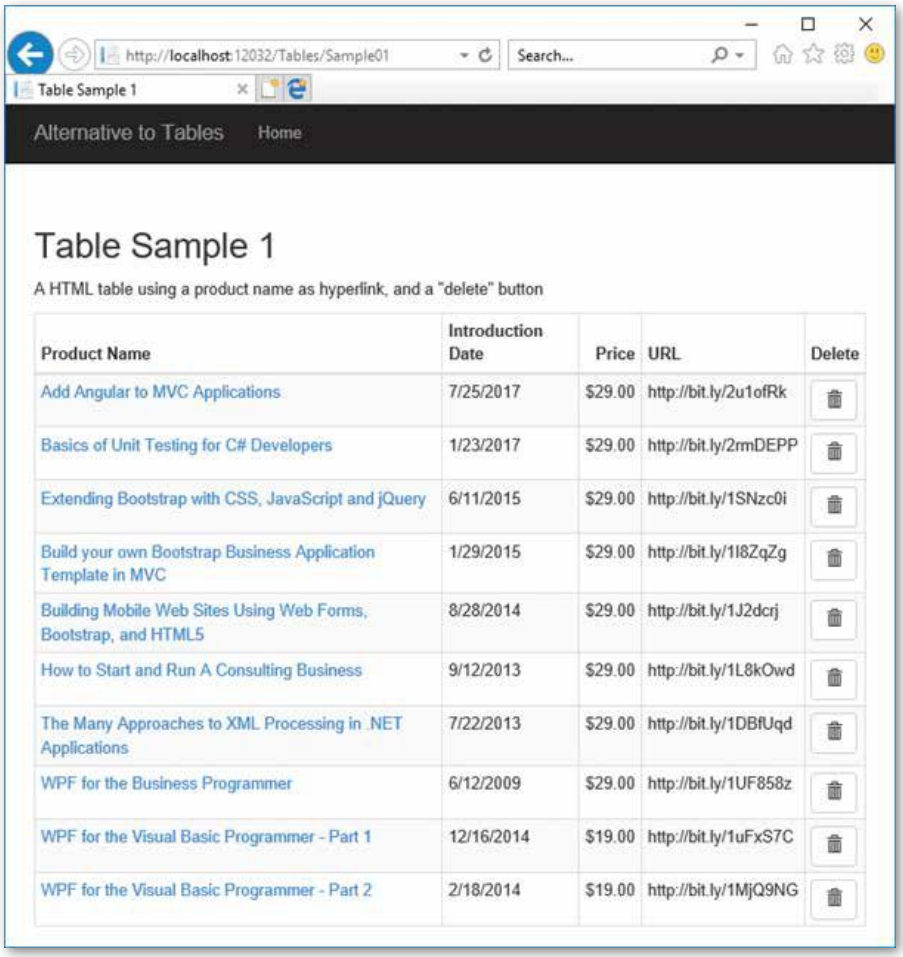


Figure 1: An HTML table rendered on a desktop browser

Listing 1: Create some mock data in the ProductManager class.

```
public class ProductManager
{
  public List<Product> Get() {
    return CreateMockData();
  }

  protected List<Product> CreateMockData() {
    List<Product> ret = new List<Product>();

    ret.Add(new Product()
    {
      ProductId = 1,
      ProductName = "Add Angular to MVC Applications",
      IntroductionDate = Convert.ToDateTime("07/25/2017"),
      Url = "http://bit.ly/2u1ofRk",
      Price = Convert.ToDecimal(29.00)
    });

    // MORE PRODUCTS CREATED HERE

    return ret;
  }
}
```

Listing 2: Create a normal HTML table as a starting point.

```
@model AlternativeTable.ProductViewModel

@{
  ViewBag.Title = "Products";
}

<h2>Products</h2>

<div class="table-responsive">
  <table class="table table-bordered
               table-condensed table-striped">
    <thead>
      <tr>
        <th>Product Name</th>
        <th>Introduction Date</th>
        <th class="text-right">Price</th>
        <th>URL</th>
        <th class="text-center">Delete</th>
      </tr>
    </thead>
    <tbody>
      @foreach (Product item in
                Model.Products) {
        <tr>
          <td>
            <a href="#"
               title="Edit Product">
              @item.ProductName
            </a>
          </td>
          <td>
            @item.IntroductionDate.ToShortDateString()
          </td>
          <td class="text-right">
            @item.Price.ToString("c")
          </td>
          <td>@item.Url</td>
          <td class="text-center">
            <a href="#"
               title="Delete Product"
               class="btn btn-sm btn-default">
              <i class="glyphicon glyphicon-trash">
              </i>
            </a>
          </td>
        </tr>
      }
    </tbody>
  </table>
</div>
```

Figure 2: The same HTML table rendered on a mobile browser

```
    <th class="text-center">Delete</th>
  </tr>
</thead>
```

You don't want to remove any of the data from your product HTML table; you just want to redistribute it into two columns. Leave the delete button in the last column, but take the code that displays the Introduction Date, Price, and URL columns and put them into the first column, as shown in the following code snippet:

```
<td>
  <p>
    <a href="#" title="Edit Product">
      @item.ProductName
    </a>
  </p>
  <strong>Intro. Date: </strong>
    @item.IntroductionDate.ToShortDateString()
    <br />
  <strong>Price: </strong>
    @item.Price.ToString("c")<br />
  <strong>URL: </strong>@item.Url
</td>
```

You should now just have two <td> elements remaining in the <tbody> of your table. If you haven't already done so, remove the attribute **class="table-responsive"** from the surrounding the <table>. Run the application, and your screen should look similar to **Figure 3**, or like **Figure 4** if you're running an emulator.

Using a Bootstrap Panel

Instead of using an HTML table, you might display a list of Bootstrap panel controls that contain your product data. In the sample shown in **Figure 5**, you see the same data, but the most important data, the product name, is

Listing 3: Use Bootstrap columns to separate labels and product data

```
<div class="row">
  <div class="col-xs-4 hidden-sm hidden-md hidden-lg">
    Intro Date
  </div>
  <div class="col-xs-4 col-md-3 hidden-xs">
    Introduction Date
  </div>
  <div class="col-xs-8 col-md-9">
    @item.IntroductionDate.ToShortDateString()
  </div>
</div>
<div class="row">
  <div class="col-xs-4 col-md-3">
    Price
  </div>
  <div class="col-xs-8 col-md-9">
    @item.Price.ToString("c")
  </div>
</div>
<div class="row">
  <div class="col-xs-4 col-md-3">
    URL
  </div>
  <div class="col-xs-8 col-md-9">
    @item.Url
  </div>
</div>
```

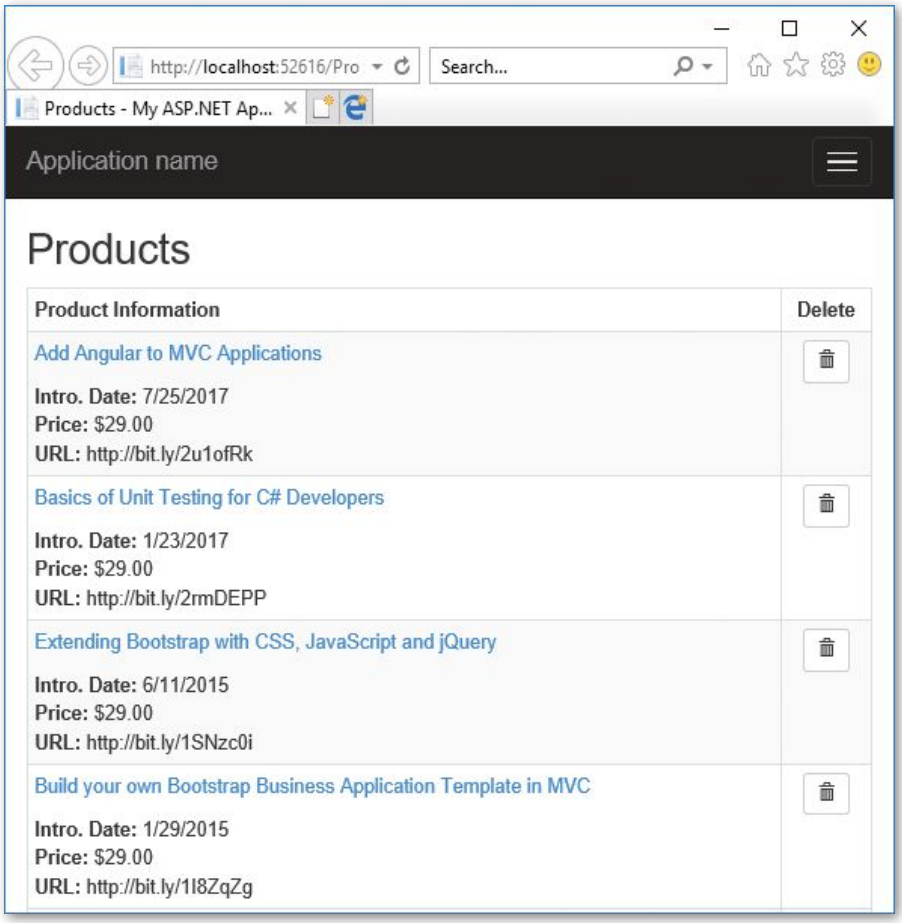


Figure 3: Reduce the table to two columns to see more within the limited space of a smart phone.

displayed in the panel header area. The other information about the product is displayed within the body of the panel. The actions you can take are displayed within the panel footer area. As you can see, this list of data looks just as good on a normal desktop browser (**Figure 5**) as on a mobile browser (**Figure 6**).

Modify the Page to Use a Bootstrap Panel

Remove all of the code below the <h2> element on your page. Write the code in the following snippet below that <h2> element. This code is just a shell of what you're going to display, but sometimes it's easier to build up the panel if you do it a little bit at a time. After typing in the code snippet, run this page to see a Bootstrap panel.

```
@foreach (Product item in Model.Products) {
  <div class="panel panel-primary">
    <div class="panel-heading">
      <h1 class="panel-title">
        @item.ProductName</h1>
    </div>
    <div class="panel-body">
      // Display product data
    </div>
    <div class="panel-footer">
       // Display actions to perform
    </div>
  </div>
}
```

Create Actions in the Panel Footer

Add the actions for Edit and Delete by using Bootstrap glyphicons within the footer of the panel. Locate the ** ** and add the following code within that element.

```
<div class="row">
  <div class="col-xs-12">
    <a href="#" title="Edit Product"
        class="btn btn-sm btn-default">
      <i class="glyphicon glyphicon-edit"></i>
    </a>
    <a href="#" title="Delete Product"
        class="btn btn-sm btn-default">
      <i class="glyphicon glyphicon-trash"></i>
    </a>
  </div>
</div>
```

Notice the use of the two glyphs for the actions that the user can take. It can sometimes be hard to click on a hyperlink with your finger on a mobile device. They can also be hard to see on a mobile device. I find using large buttons with a graphic gives the user a nice big target to hit with a finger.

Build the Panel Body Using Rows and Columns

Fill in the body of the panel with the rest of the product data. The code in **Listing 3** uses Bootstrap row and column classes to display the appropriate product data within the Bootstrap body of the panel.

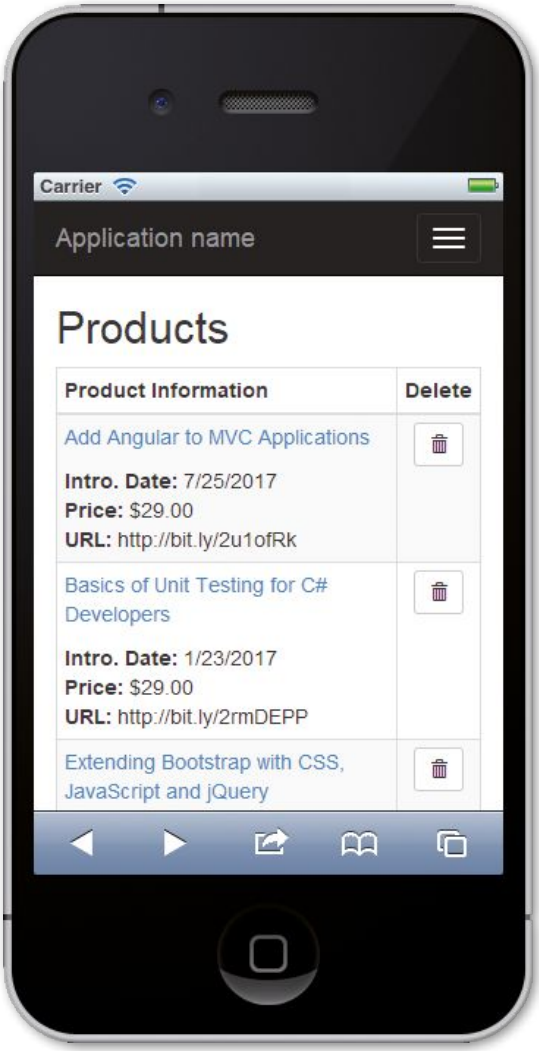


Figure 4: Reducing your table to two columns makes it fit well on a smart phone.

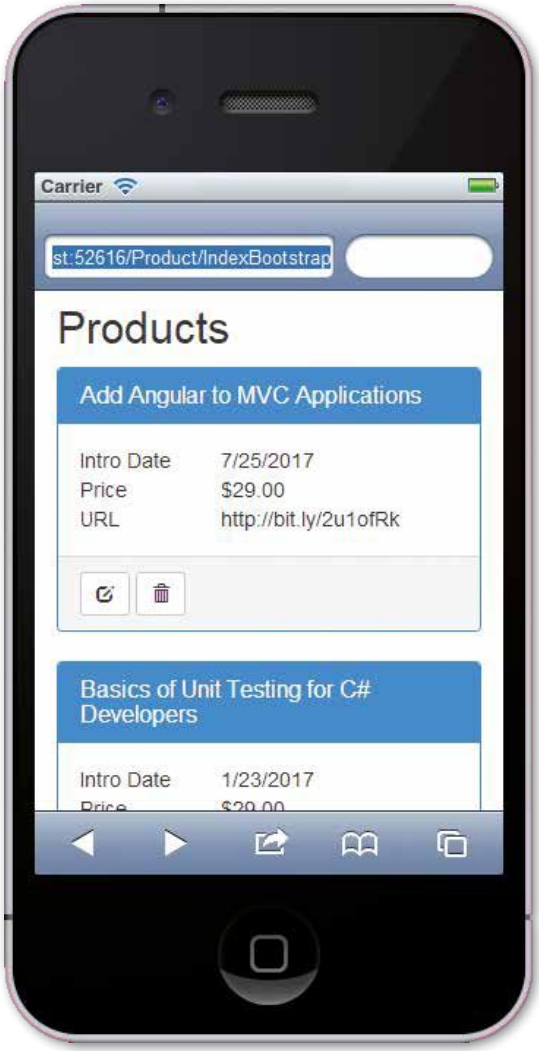


Figure 5: Display product data using the Bootstrap panel classes

Let's examine the code in **Listing 3** to understand how it was put together. The column width is modified depending on the size of the browser. For anything that's a medium resolution and above, the col-md-3 class is used for the first column and a col-md-9 for the second column. As soon as a mobile device is detected, the col-xs-4 class is used for the first column and col-xs-8 for the second. If you look at the Introduction Date field, you also notice that the words used in the label change depending on the size of the browser. This is one of the great things about using Bootstrap: the ability to hide and display things using simple CSS classes.

Detect a Mobile Device and Switch Views

You've made some great changes to your Product page to make it look good on a mobile device. However, sometimes, your users insist on seeing tabular data. There's no reason you can't have the best of both worlds. When browsing on a desktop browser, you can have a page that looks like **Figure 1**, but when viewing on a mobile device you can have the page that looks like **Figure 6**.

To accomplish this, you're going to use partial pages and some basic device detection code, as illustrated in **Figure 7**. You're going to write the device detection code in a minute, but first open the ProductViewModel class and add a Boolean property named IsMobileDevice. This value is going to be set by the device detection code you write. This property helps your Index.cshtml page decide which partial page to display.

```
public bool IsMobileDevice { get; set; }
```

Create two .cshtml pages: one for the code I had you save earlier in this article, and one for the current code to display the product data. Right mouse-click on the Product folder and select **Add** > **MVC 5 Partial Page... (Razor)** from the menu. Name one of these partial pages **_ProductTable.cshtml** and the other **_ProductPanel.cshtml**. At the top of each of these partial pages, add the following line of code.

```
@model AlternativeTable.ProductViewModel
```

Create the partial page for the desktop browser by getting the Razor code you saved earlier with the original

Sample Code

You can download the sample code for this article by visiting my website at http://www.pdsa.com/downloads. Select PDSA Articles, and then select "CODE Magazine: Eliminate HTML Tables for Better Mobile Web Apps" from the drop-down list. You can also download them from the CODE Magazine page associated with this article.

Listing 4: A simple device detection method.

```
public bool IsMobileDevice()
{
  bool ret = Request.Browser.IsMobileDevice;
  string userAgent = Request.UserAgent;

  if (!ret) {
    // Use regular expression
    Regex b = new Regex(@"(android|bb\d+|meego)
    .+mobile|avantgo|bada\/|blackberry|
    blazer|compal|elaine|fennec|hiptop|
    iemobile|ip(hone|od)|iris|kindle|lge
    |maemo|midp|mmp|mobile.+firefox|
    netfront|opera m(ob|in)i|palm(os)
    ?|phone|p(ixi|re)\/|plucker|
    pocket|psp|series(4|6)0|symbian|
    treo|up\.(browser|link)|vodafone|
    wap|windows ce|xda|xiino",
      RegexOptions.IgnoreCase |
      RegexOptions.Multiline);

    ret = b.IsMatch(userAgent);

    // Check user agent for certain words
    ret = !ret
          ? userAgent.ToLower().Contains("iphone")
          : true;
    ret = !ret
          ? userAgent.ToLower().Contains("android")
          : true;
    // etc.
  }

  return ret;
}
```

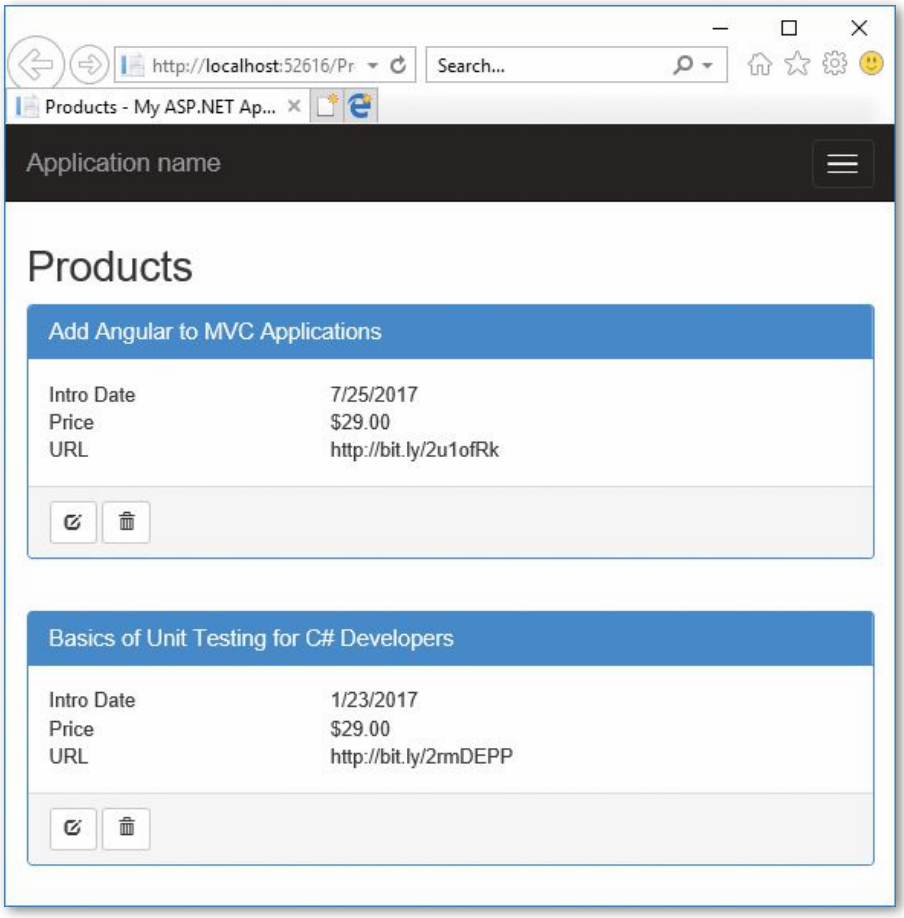


Figure 6: The Bootstrap panel classes look very good on a mobile device.

HTML table. Copy the code below the <h2> into the _ProductTable.cshtml file. Create the partial page for the mobile browser by cutting out the code below the <h2> in the Index.cshtml file and pasting that into the _ProductPanel.cshtml file. Now modify the Index.cshtml file to look like this next snippet.

```
@model AlternativeTable.ProductViewModel

@{
  ViewBag.Title = "Products";
}

<h2>Products</h2>

@if (Model.IsMobileDevice) {
  @Html.Partial("_ProductPanel", Model)
}
else {
  @Html.Partial("_ProductTable", Model)
}
```

Run your application in your desktop browser and ensure that you see the full HTML table. Next, stop your application and set the **IsMobileDevice** property to a true value. Re-run the application and ensure that you see the Bootstrap panel.

Simple Device Detection Code

There are many methods to detect what kind of browser is running the method in your controller, ranging from very simple to very complex. You can even purchase a subscription to a service for device detection. In this article, you're going to write a simple method to detect a mobile browser. I've found that this method works for most mobile browsers on the market today.

You're going to want to use this device detection method on many controllers, not just the Product controller you created in this sample. Create a controller base class from which all your controllers can inherit. Right mouse-click on the \Controllers folder and add a new class named **AppBaseController**.

```
using System.Web.Mvc;
using System.Text.RegularExpressions;

namespace AlternativeTable.Controllers
{
  public class AppBaseController : Controller
  {
  }
}
```

Open the ProductController.cs file and change this class to inherit from this new base controller class.

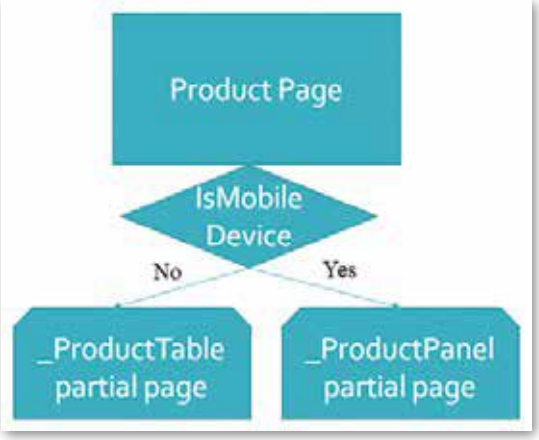


Figure 7: Based on device detection, you display one partial page or another.

```
public class ProductController :
  AppBaseController
{
   // REST OF YOUR CODE HERE
}
```

Next, add a new method named IsMobileDevice() to your AppBaseController class. This method looks at a couple of properties of the **Request** object to determine whether the browser running the code is a mobile browser. **Listing 4** contains the complete code for the IsMobileDevice() method.

The IsMobileDevice() method looks at the **Request.Browser.IsMobileDevice** property to see if MVC has detected a mobile browser. If this value is not true, you then check a string against the Request.UserAgent to see if you can find some certain keywords. If any of these keywords exist, then you have a mobile device.

If the regular expression fails, check the user agent string to see if it contains "iphone" or "android". You can add additional checks for other types of mobile devices, such as a blackberry. Instead of hard-coding these values in this method, you might create a couple of key/value pairs in the <appSettings> section of your Web.config file to hold the string to check for the regular expression, and for each of the phone keywords.

Product Controller

Now that you have a device detection method in place, call this method and set the **IsMobileDevice** property you added to your ProductViewModel class. Open the ProductController.cs file and modify the Index() method in the controller to call this method and set that property as shown in the following code snippet.

```
public ActionResult Index()
{
  ProductViewModel vm = new ProductViewModel();

  vm.IsMobileDevice = IsMobileDevice();
  vm.LoadProducts();

  return View(vm);
}
```

Run the application using a desktop browser to ensure that your code detects a normal browser and returns the full HTML table. Next, try running your application from an emulator. If you don't have an emulator, you can change the user agent string using the F12 tools in your browser. You may have to search the Internet for how to change the agent string using your unique browser, as they each work a little differently. If you use Chrome, it's very easy to set the user agent string; if you use IE 11, you have to turn off the compatibility settings in order to get the user agent string to submit correctly.

Summary

Every Web developer should create applications with mobile in mind. More people browse sites with mobile devices than with desktop browsers now. Your Web application should respond accordingly. Eliminating HTML tables goes a long way to making your users happier when browsing data on their smart phones. Adding device detection, and modifying your pages based on the browser type increases your development time, but can keep your users happy.

Paul D. Sheriff

Bots

Now that I have your attention, what does the word "bots" remind you of? Does it remind you of things such as natural language recognition? Artificial Intelligence? Computers stealing our jobs? Singularity, World War X, becoming pets to our computer overlords, the end of humanity as you know it? Calm down! Bots are nothing like that. And no, the computer didn't bribe me with a piece of cheese to say this.

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. You can find more about his training at http://www.winsmarts.com/training.aspx.

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

As computers have grown in complexity, our way of interacting with the computer has also changed. We used to interact using printouts and punch cards. Then some genius invented a CRT screen with great contrast ratio, and we had a terminal and a command prompt. How convenient was that; I could type in my commands using a text-based interface, and the computer responded. No more shuffling around those cards. The problem was remembering all those commands, so the developers of complex programs came up with menu-driven interfaces. You started the program and it presented you with a number of options to pick from. You picked an option and went to another screen full of options. Those were the days. Then came the GUI. The mouse was a particularly amazing invention. You could drag-and-drop things, and open and close windows. Most recently, we have touch screens. Who needs a mouse when you have ten styli attached to your hands, right?

Bots are simply the next evolution in how we interact with a computer.

A bot is an app that lets users interact with the app in a conversational way. The Microsoft Bot Framework is a framework that helps you write bots.

Conversational User Interface

Yes, I'm sorry to puncture your enthusiasm; this article won't mean World War X and the end of humanity. I'm simply describing a new way of interacting with a computer: through conversations.

You, the human, interact with a program via a conversation. Let's say that you start with "Hello."

The bot receives this message and can interpret your intent using numerous mechanisms.

- It can try to do an exact match; this is nothing but simple string matching.
- It can try to recognize patterns; for instance, if the user's input starts with "Hello," this must be the start of a conversation.
- It may use regex to do more complex pattern recognition.
- Or it may use LUIS (Language Understanding Intelligent Service), where the app literally tries to understand the user's intent. For instance, "What is the weather outside," or "Do I need an umbrella," or "How cold is it outside," all mean the same thing, as far as the bot's concerned.

Once the bot (your program), understands the user's intent, it can respond with a new set of questions, possible actions, or results. Results could be speech, text, a list, or anything similar.

The good news is that you don't need to be an AI expert to start writing your first bot. In fact, to understand the nuts and bolts of how the Bot Framework works, you need to know nothing more than exact string matching. And the other pieces logically fit in to place.

> To understand the nuts and bolts of how the Bot Framework works, you need to know nothing more than exact string matching.

Before I dive into code and a functional bot, there are some important concepts to understand.

Good Bot Design

Users have many ways of getting the information they need, but they have very little time or attention. When they want to know about the weather, they could go to a website, pick up a phone, or say "hey Siri." They'll probably pick the route that's the least effort and still gets the answer they need.

Keep this in mind when designing your bot. For starters, you might make a single catch-all bot, the kind that will end the human race. That's probably not a good idea. Your bot should be specific to a function, and even within that function, it should narrow down the choices and the user's intent to something very specific. Here are some guidelines:

- Don't try to make the bot too smart. A bot that tries to do too much is not the best candidate because it will fail to recognize the user's intent. For instance, if you said, "search for Sahil," should it return a YouTube video or a GitHub repo? How about, "search for the last thing Sahil published." What exactly are you talking about here?
- Don't try to be cute! People use bots that are efficient and to the point. Don't pretend to be human; you aren't good at it. Jokes can be annoying, especially if they get in the way of getting things done. When I ask for weather, tell me, "It's 55F outside." There's no need to follow that up with a cute rejoinder.
- Voice and natural language recognition aren't quite there yet. Don't get me wrong; they can be quite effective. For instance, when the user says a phrase, you could use Microsoft cognitive services to recognize the user and perhaps load their non-secure user

VENDORS: ADD A REVENUE STREAM BY OFFERING ESCROW TO YOUR CUSTOMERS!

CAN YOU TAKE THE RISK ?

Less than $1 per day!

Affordable High-Tech Digital Escrow

Tower 48 is the most advanced and affordable digital escrow solution available. Designed and built specifically for software and other digital assets, Tower 48 makes escrow inexpensive and hassle free. Better yet, as a vendor, you can turn escrow into a service you offer to your customers and create a new revenue stream for yourself.

Regardless of whether you are a vendor who wants to offer this service to their customers, or whether you are a customer looking for extra protection, visit our web site to start a free and hassle-free trial account or to learn more about our services and digital escrow in general!

Visit **www.Tower48.com** for more information!

TOWER 48

profile. It needs to be non-secure because the voice could be recorded. However, when I say, "recognize speech," how are you sure that I wasn't saying "wreck a nice beach?" The reality is that we humans are very good at voice recognition because we pair the meaning with context. So first establish context, and then try using these fancy tools. And ask any married couple, even with context, we humans sometimes mess up understanding each other. Surely the bots can't be perfect either. Plan for imperfection.

- Keep it simple and expose it to as many channels as possible. The Microsoft Bot Framework can easily allow you to expose your bot functionality across numerous channels, such as Skype, SMS, email, Microsoft Teams, Facebook, and more.
- Narrow the problem down, starting with your welcome message. For instance, imagine a bot that responds to my hello with "Hello Sahil," versus another bot that says "Hello Sahil. I can help you 1) Book an Air Ticket, 2) Book a hotel, 3) Book a car. Which would you like to do?" Which of the two do you think leaves me less confused?
- Break down your complex problems into simpler steps: these steps are your dialogs. For instance, the process of booking an air ticket could first start by asking "When are you leaving," followed by "Where are you leaving from," etc. You're effectively stepping through dialog after dialog, only in a conversation.
- Recognize that humans handle and expect exceptions. For instance, right in the middle of my conversation, the human could say, "can you check my calendar for me?" This means that your programming paradigm needs to support:
    - Start a new conversation.
    - Replace the current conversation.
    - Add to the existing conversation.

As you can see, it boils down to a lot of common sense. If I had to give you three tenets, they would be:

- Keep it simple; narrow down the problem.
- Don't be annoying.
- Be useful; get to the information that the user needs fast.

We're still in the infancy of bot design and a lot of best practices are still emerging. We're still not sure whether we find Siri's jokes annoying or useful. Luckily, the Bot Builder SDK and the Microsoft Bot Framework encompass concepts that let you model a bot. You could still design a bad bot, but then, you could, I hope, design a good bot too. Let's get a little bit more technical now.

The Bot Builder SDK

The BOT builder SDK from Microsoft allows you to build bots using the Bot Framework. It models all the concepts necessary, such as dialogs, channels, state, interaction with LUIS etc., to help you build compelling bots. Your bot may or may not be hosted in Azure, but hosting it in Azure is easier. Currently the Bot Builder SDK is available for .NET as a NuGet package and for NodeJS as an NPM module, or, you can simply interact with it using REST APIs. There may be additional NuGet packages or node modules specific to channels.

Dialogs

Typically, you organize your interaction with a bot using conversations. These conversations are modelled as one or more dialogs. Dialogs contain prompts and waterfall steps, which is simply a number of sequential dialogs that the user goes through in logical progression, and at the end of that progression, the "waterfall" returns the entire resultset.

Let's look at a conversation organized into dialogs.

- Usually, you start with a default dialog. For instance, "Hello Sahil; how may I help you?"
- Based on some trigger, such as a word match like "Help," you may jump to an alternative dialog. For instance, "Help, I'm lost," to which the bot can respond, "Sahil, I can help you with, 1) Book a flight, 2) Book a hotel room, 3) Book a car?"
- One dialog can redirect to another dialog. This redirection may be a logical "next" or it can be based on a condition. For instance, when using a default flow, "When are you flying out" may be followed by "Where are you flying to." Based on a condition might be, "I want to book a hotel," which might result in a dialog #2, which is about hotel reservations.
- One dialog can also redirect to another dialog based on an interruption or an action. When you jump based on interruptions and start a new dialog flow, you may resume back with the original dialog after the interruption is complete. For instance, in the middle of booking my travel plans, I may say, "When am I flying out?" which then may have the bot respond with "April 20th, 2018," or "You have three flights in the next two months; which one would you like me to read out?" What's important here is that you want to remember the current dialog chain, because after the user knows when they're flying out, they'll want to return to the original dialog flow.
- Sometimes the redirection may cause the entire dialog redirection chain to start over, for instance, if the user says, "I'm confused, let's start over."
- Dialogs usually follow a waterfall pattern, which guides the user through a series of steps or prompts.

Each one of these concepts—a dialog, a trigger, a dialog stack, a waterfall, actions/interruptions, replace dialog, and resume dialog—are modelled in the Bot Builder SDK.

State

Bots run as a Web service. This Web service is stateless. It could run over many nodes, and it might even be serverless. It could be something as simple as an Azure function. So the bot itself doesn't store any state; however, no self-respecting program—or should I say bot—could function without state. Like most newer applications, state is externalized and your program is stateless. State in the Bot Builder SDK is based on a provider model, and out of the box, it supports three kinds of state:

- In-memory storage: This is great for development purposes, but remember, because it's "in memory," this kind of bot won't scale across multiple nodes.
- Azure Table storage
- Azure Cosmos DB

You can also author a custom state provider using the Bot Builder SDK Azure Extensions (https://github.com/Microsoft/BotBuilder-Azure). The good news is that no matter what state provider you choose, as long as you stick with the well-defined API, your program doesn't change. The state provider is simply plug-and-play. It's important that you stick with the various properties and methods that the Bot Builder SDK exposes, and not try to reinvent the wheel.

The Bot Builder SDK for NodeJS session object exposed the following properties for state data:

- **userData:** The state that's scoped to the user and is accessible over multiple conversations. For instance, if I asked to book a flight, userData is a great place to store this information because in a subsequent conversation, I could easily change the flight.
- **privateConversationData:** This state is for the user in this particular conversation. For instance, if I'm booking a flight and I say "forget about it," I could simply call **session.endConversation**, and forget everything specific to me in this current conversation with the bot.
- **conversationData:** This is specific to a conversation, but is shared among all users in the conversation. However, this state is also cleared when the conversation is ended with the **endConversation** command. For instance, in a Microsoft Team's chat where multiple users are querying about a JIRA issue, clearing the conversation should effectively reset the state. For instance, three developers and one bot enter into a conversation. Developer 1 says, "open JIRA issue 331." The bot responds with, "JIRA issue 331: modify styles to reflect updated branding, is now open." Developer 2 can then ask, "What Git branches were created for this issue," etc. This could continue until a new issue is opened, which opens a new conversation. What's important is that in the second message where Developer 2 says "this issue," the bot clearly understands that "this" means JIRA issue 331, which is stored in conversationData.
- **dialogData:** This contains data specific to the current dialog only.

The Bot Builder SDK for .NET exposes similar capabilities via Getter and Setter methods. The method names and properties may be different, but the concepts are the same.

Channels

Bots are, put simply, programs that you interact with via conversations. And just as we humans can have conversations on many mediums, or let's say channels, bots can interact via multiple channels. Examples of channels are a simple website that hosts a chat-based channel, like Skype or Cortana. As you may have guessed, Cortana may have some peculiarities, such as speech. Microsoft Teams is yet another channel, which introduces its own unique set of peculiarities and features. For instance, a Teams-based bot can do 1:1 conversations or participate in a group chat.

It can also send notifications to a user. Imagine if I had a helper bot that I could tell, "@documentWatcher, please alert me when a new document is created with the phrase "pay raise for Sahil." As you may have guessed, this action that I specified to my Microsoft Team's bot is handled by a Web service. This Web service is nothing but an app, as far as Microsoft Graph is concerned. Just like any app, it has permissions. Based on those permissions, it could execute a search and keep watching for documents matching my specified phrase. And when such a phrase match occurs, it can notify me, which is simply sending a message to the user.

The phrase match doesn't even have to be exact; using LUIS, you can add natural language intelligence, even in the matched documents. For instance, "Should we give Sahil more money" is somewhat close to "Give Sahil a pay raise." I should know about that too, right? What about "Should we fire Sahil?" Yes, with natural language recognition, I could write logic around that too. It's quite incredible. I'm sure you're beginning to see the value and power of mixing bots in your various channels already.

Developing and Testing Your Bot

There's one last thing you need to know before you dive into code, and that's how you go about developing and testing your bot. As I mentioned, the bot is just a Web service. But it's a Web service that the surface you're testing, be it Microsoft Teams, a Web-hosted chat, or something else, should be able to communicate with. There are three options here.

The first and the simplest option is to use a cross-platform app written in Electron that's called as the Bot Framework emulator. You can download this emulator from https://github.com/Microsoft/BotFramework-Emulator. This emulator allows you to write and test your bot without exposing anything to the Internet.

The other two options require exposing your development endpoint to the Internet. Although this may sound scary, the reality is that in this cloud-friendly world we live in, Azure-based services frequently need to talk to dev endpoints. Naturally, easier solutions have emerged to this problem. My favorite is ngrok, a simple exe that exposes your dev endpoint to an HTTP or HTTPS URL on the Internet that ends in ngrok.io. The only consideration here is that the endpoint changes every time you run ngrok. Luckily, the bot registrations in Azure make it easy to change the endpoint.

The other option is to expose the endpoint by hosting it on the Internet. I like to use Azure websites because even though the endpoint is exposed to the Internet, I can still debug the code in dev mode. Yes, debugging an Azure site is slower than debugging something locally, so my preferred approach is ngrok. I just run it, minimize it, and forget about it. I can restart my Web service endpoint as many times as I wish. And if I have to update my endpoint URL in my Azure bot registration once a day, it's not that big a deal.

The Hello World Bot

So far, there's been lots of talk; it's time to see some code. Let's start by authoring a simple "Hello World" bot. I'll use NodeJS, but everything I show here can also be done using .NET.

Listing 1: The Hello World Bot package.json

```
{
  "name": "simplebot",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "node index.js"
  },
  "dependencies": {
    "botbuilder": "^3.13.1",
    "restify": "^6.3.4"
  },
  "devDependencies": {
    "@types/restify":"^5.0.6"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Listing 2: Hello World Bot

```
const builder = require('botbuilder');
const restify = require('restify');

const connector = new builder.ChatConnector();
const bot = new builder.UniversalBot(
    connector,
    [
        (session) => {
            session.send('Hello world!');
        }
    ]
).set('storage', new builder.MemoryBotStorage());

const server = restify.createServer();
server.post('/api/messages', connector.listen());
server.listen(3978);
```

To start with, like any other node-based project, create a package.json file. In this package.json file, add the code shown in **Listing 1**.

As can be seen in **Listing 1**, in addition to referencing the botbuilder npm package, I also use restify to expose it as a Web service. Also, the start command uses node to run index.js, the code for which can be seen in **Listing 2**.

Listing 2 shows the simplest bot you can make. No matter what message you send it, it replies with "Hello World." As can be seen, I'm using builder.UniversalBot, which accepts two parameters. The first is the connector on which I wish to expose the bot. The second is an array of dialogs, of which I have only one. Usually these are functions, and the first argument for it is **session**. Session allows me to get a lot of valuable information. For instance, what the user said last can also be grabbed off of the session object. I'm keeping it simple and simply replying with "Hello World" no matter what the user said.

Let's test the bot next. Download the Bot Framework emulator and run it. Also in your project, run npm install and npm start. Once your project is running, switch over to

Qualified, Professional Staffing When You Need It

CODE Staffing provides professional software developers to augment your development team, using your technological requirements and goals. Whether on-site or remote, we match our extensive network of developers to your specific requirements. With CODE Staffing, you not only get the resources you need, you also gain a direct pipeline to our entire team of CODE experts for questions and advice. Trust our proven vetting process, and let us put our CODE Developer Network to work, for you!

Contact CODE Staffing today for your free Needs Analysis.

Helping Companies Build Better Software Since 1993

www.codemag.com/staffing
832-717-4445 ext. 9 ● info@codemag.com

CODE STAFFING

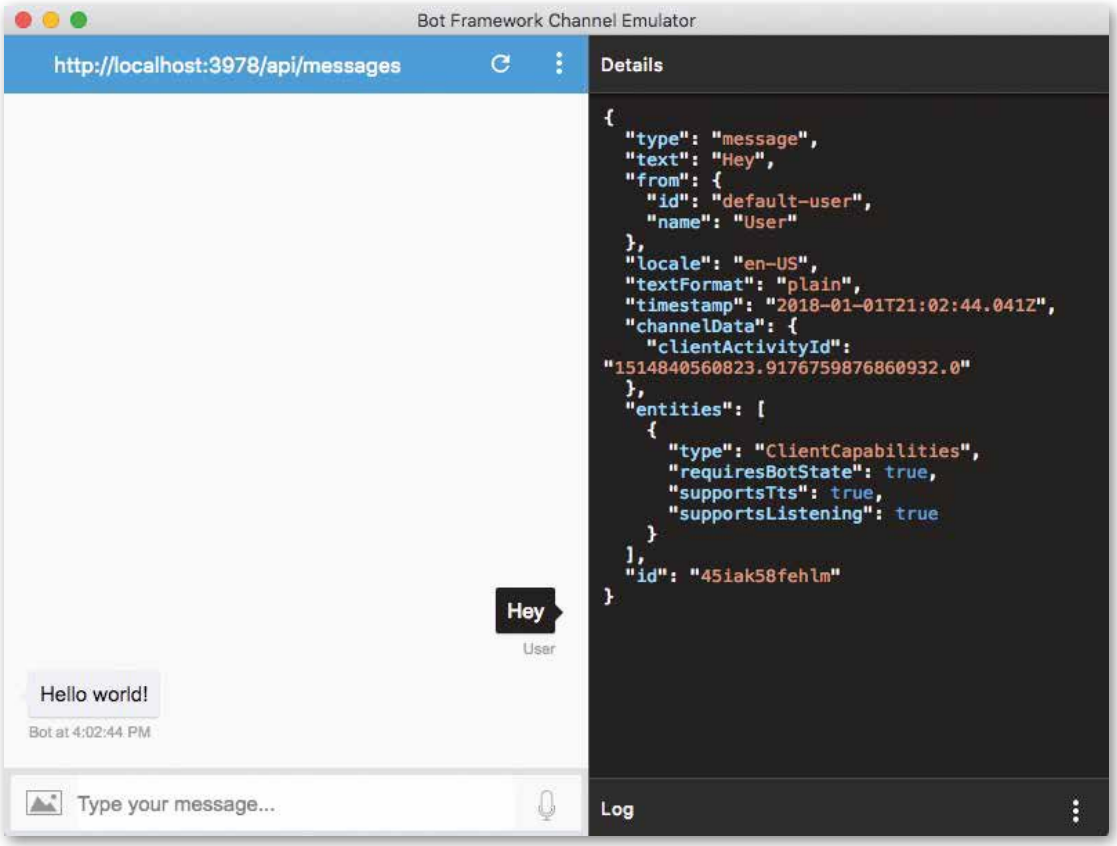


Figure 1: The Hello World Bot running

the Bot Framework emulator, and connect to http://localhost:3978/api/messages. You don't need a Microsoft App ID or App Password yet; these become relevant once you start deploying the app in Azure or using it in connectors.

Once connected, go ahead and type any message to your bot; you should see the bot respond with "Hello World!" as shown in **Figure 1**. You can also see in **Figure 1** that I can click on any message and see the details of that message as JSON in the right-hand pane.

Writing a Bot for Microsoft Teams

The great thing about the Bot Framework is that one bot can be exposed to multiple channels. One such channel is T. But what fun is it to expose a simple Hello World bot to T? Well, the fact that you're running inside Microsoft T should pose some interesting opportunities.

A Microsoft Teams bot can:

- Have 1:1 chats, where the user messages the bot just like they would message another user.
- Have chats inside a channel, so the bot becomes a member of the Team, and multiple users can talk to the bot.
- Notify a user by @mentioning them or sending them a notification
- Fully interact inside the Teams "context." It can do things like get the user's profile, get a list of channels in a Team, get members of a Team, etc.
- Be granted permissions in Microsoft Graph just like you'd grant permissions to any other Graph application. This means that the bot can act as a graph application, and query for things such as mail, calendar, documents, and much more.

An Important Note

At the time of writing this article, there are two ways to register a bot. You can go to https://dev.botframework.com/ and click "Create a bot or skill," which is the equivalent of logging into your Azure portal and provisioning a new bot. Or you can go to https://dev.botframework.com/bots/new. When you use the second link, you get the option of migrating the bot to Azure. Down the road, we'll most likely create bots only in the Azure portal. Right now, Teams bots are more reliable if you create them directly in the https://dev.botframework.com/bots/new URL. I've noticed strange behavior, such as a newly registered bot in Azure being responsive on every channel except Teams. I'm sure these growing pains and kinks will be sorted out before Azure is the only way to register bots. But until then, I'd suggest creating them using the https://dev.botframework.com/bots/new URL. I'll mention both Azure and non-Azure instructions below.

Registering a New Bot for Microsoft Teams

Visit https://dev.botframework.com/bots/new, and sign in with your Office 365 credentials. If you intend to create the bot using the dev.botframework.com URL, you'll still sign in using your Office 365 credentials, but this credential must also have an Azure subscription associated with it.

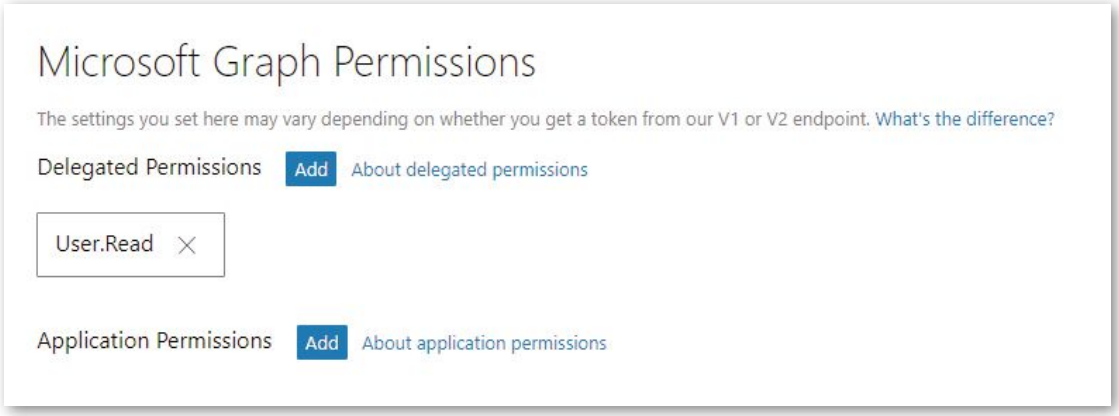


Figure 2: Permissions for the Team bot

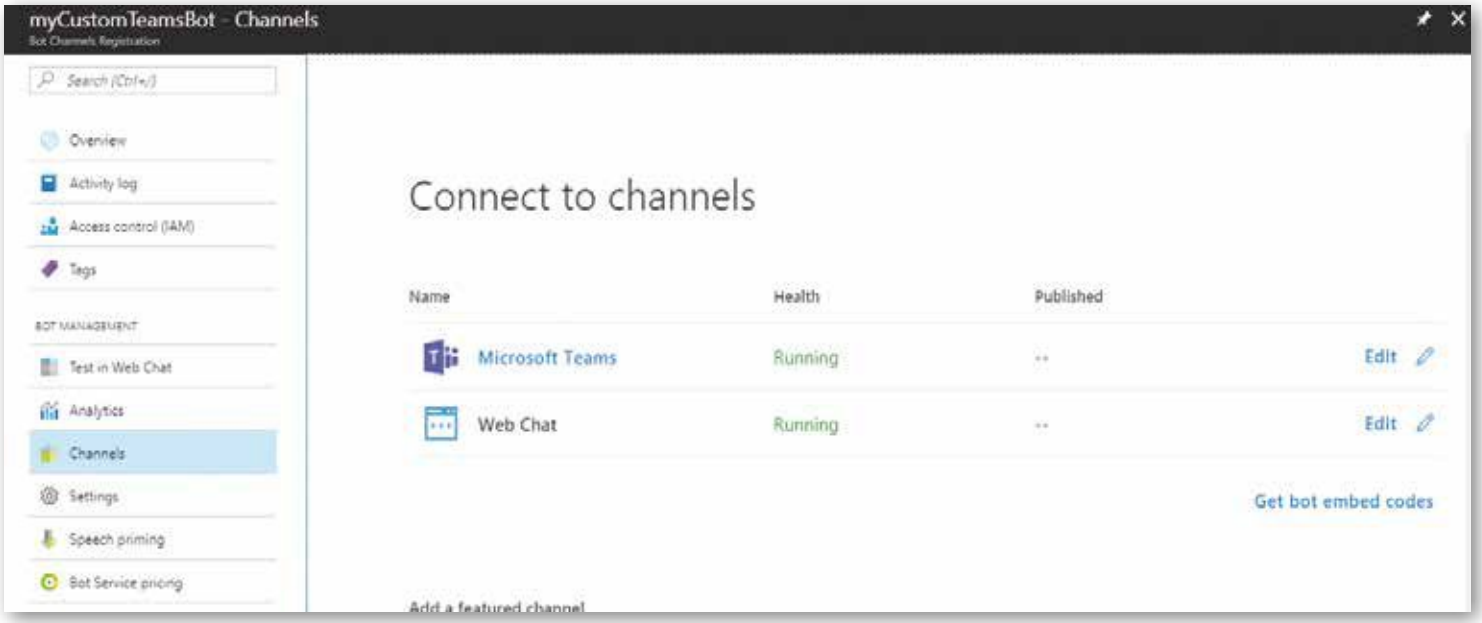


Figure 3: Adding the Teams channel: the new user experience

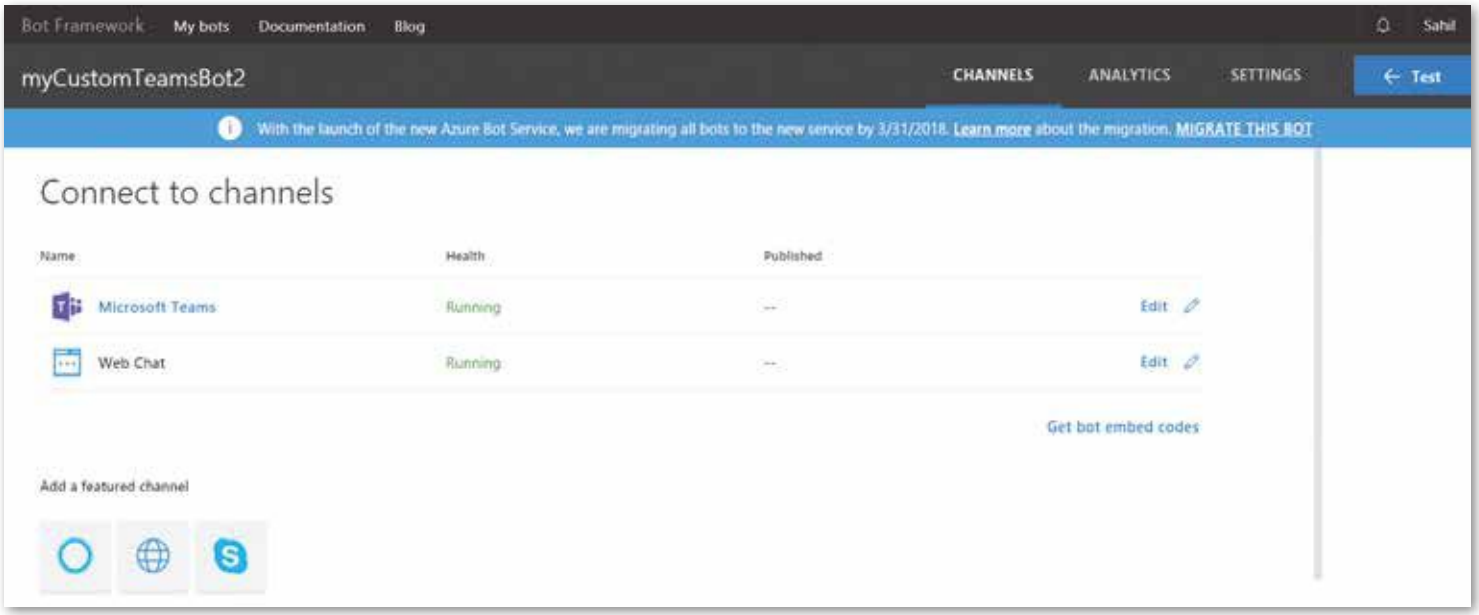


Figure 4: Adding the Teams channel: the old user experience

Registering a bot is a matter of filling out a simple form. You need to specify things like the display name, the @botHandle, a description, etc. The interesting thing is the messaging endpoint. This messaging endpoint is what you will expose to the Internet either via ngrok or an Azure website. You haven't exposed anything yet, so just type https://something there for now.

You'll also need to generate a Microsoft AppID and Password. Behind the scenes, you're simply registering the bot as an app using the v2 app model. When you generate the AppID and Password, note it down somewhere. You'll need it shortly.

Also, under the "Microsoft Graph Permissions" section, grant it the "User.Read" permission, as shown in **Figure 2**.

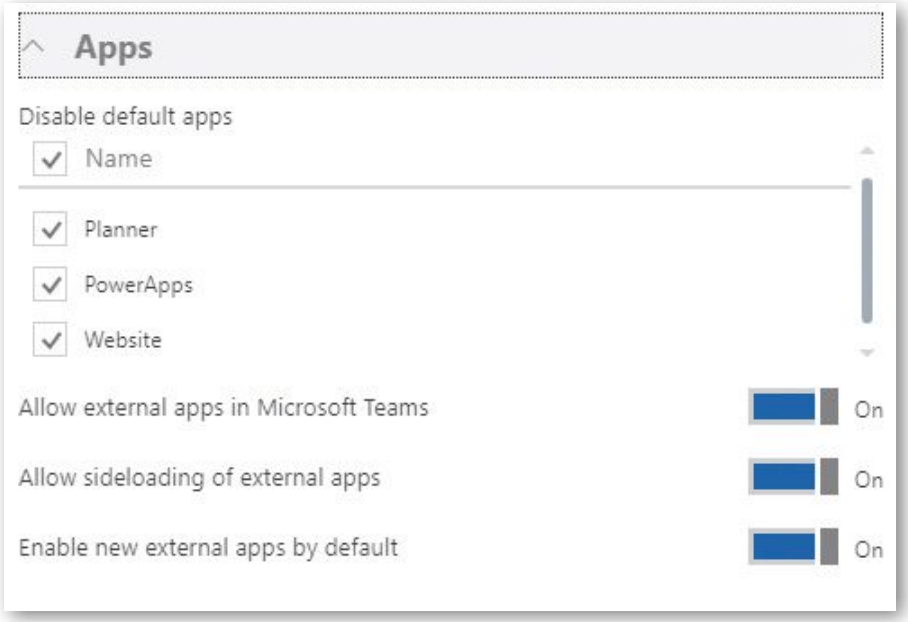


Figure 5: Enable sideloading of apps

Configure Channels for Your Bot

Here, the instructions become different based on whether this is a bot you registered using the old URL directly in Microsoft Teams, or via Azure. Visit https://dev.botframework.com/bots to see the list of bots you've registered. Clicking on the bot should take you either to the Azure portal (if you use the Azure/new way of registering) or to a different page if you used the old way of registering.

If you're in the Azure portal, look for "channels" and choose to add the "Microsoft Teams" channel, as can be seen in **Figure 3**. Alternatively, if you're using the old way of registering your bot, you'll be presented with a different user experience, as can be seen in **Figure 4**.

One last thing before we leave the bot registration page. Under the "Settings area" in the top navigation bar for the old registration portal, and left side navigation area

Listing 3: Our Teams-ready bot

```
const builder = require('botbuilder');
const builderTeams = require('botbuilder-teams');
const restify = require('restify');
var githubClient = require('./github-client.js');

const connector = new builderTeams.TeamsChatConnector(
    {
        appId: "removed",
        appPassword: "removed"
    });

const bot = new builder.UniversalBot(connector)
    .set('storage', new builder.MemoryBotStorage());

const dialog = new builder.IntentDialog();

dialog.matches(/^getchannels/i, [
    function (session, args, next) {
        console.log(session);
        var conversationId =
   session.message.address.conversation.id;
        connector.fetchChannelList(
            session.message.address.serviceUrl,
            session.message.sourceEvent.team.id,
            (err, result) => {
                if (err) {
                    session.endDialog(JSON.stringify(err));
                }
                else {
                    session.send('Here are the matching channels');
                    session.endDialog(JSON.stringify(result));
                }
            }
        );
    }
]);

dialog.matches(/^aboutme/i, [
    function (session, args, next) {
        console.log(session);
        var conversationId =
    session.message.address.conversation.id;
        connector.fetchMembers(
          session.message.address.serviceUrl,
          conversationId,
          (err, result) => {
            if (err) {
              session.endDialog(JSON.stringify(err));
            }
            else {
              session.endDialog(JSON.stringify(result));
            }
          }
      );
    }
]);

bot.use(new builderTeams.StripBotAtMentions());

bot.dialog('/', dialog);

const server = restify.createServer();
server.post('/api/messages', connector.listen());
server.listen(3978);
```

under channels in the Azure portal, you'll see a place called "Configuration" and "messaging endpoint". Just take a note of that; You'll need to modify the value there once you expose the bot on the Internet using ngrok.

Writing the Teams-ready Bot

In the nodejs-based project, remove the reference to **botbuilder**, and instead add a reference to **botbuilder-teams**. I'm using version 0.1.7. Note that botbuilder-teams include a reference to botbuilder, so there's no need to add it explicitly. There are no other changes in package.json.

Inside index.js, write the code as shown in **Listing 3**. As can be seen in **Listing 3**, the code is strangely familiar. It is, after all, a bot! But, this time around, I'm using dialogs. And the two dialogs I'm using are triggered based on a regex match. One is triggered when the user writes "getchannels" to the bot. The second is triggered when the user writes "aboutme" to the bot.

It's very important to realize that the "aboutme" action can work in either a personal chat or in a group chat in a channel. However, getchannels requires you to provide a Team ID, and the only way to get the Team ID currently is from the session's context. In other words, in order to test "aboutme," I can simply reference the bot in a personal chat. I don't need to package it up as a zip file and add it to Teams. To test it inside the context of a Team, I need to upload the bot into Teams as a zip file. I'll leave that for another day; for now, let's just see how to test the "aboutme" action.

Testing Your Bot

There are two things you need to do in order to test your bot.

First, you need to enable sideloading of bots in Microsoft Teams. Log in as tenant administrator and go to the admin area. Under Settings\services and add-ins, look for Microsoft Teams. Ensure that sideloading of external apps is enabled under the Apps section, as shown in **Figure 5**.

It's worth mentioning that most administrators feel itchy turning this capability on. I advise you to do this in a developer tenant and not in the production tenant. I wish we could turn this capability on as a group-by-group basis, but we can't. Not today at least.

The second thing is to simply test the bot out.

Go ahead and run the bot. This is a matter of doing **npm prune** (to remove botbuilder), **npm install** (to add botbuilder-Teams), and **npm start**, to run the bot.

Once your bot's running, download **ngrok** from ngrok.io, and run **ngrok http 3978**, where 3978 is the port your bot runs on locally.

Ngrok should give you two URLs: an HTTP URL and an HTTPS URL. Essentially, ngrok exposes your port 3978 on the Internet via a duplex outbound TCP channel. If that doesn't impress your mom, I don't know what will.

Once you have this ngrok URL handy, go ahead and update it under the Messaging Endpoint area of your bot registration. Mine looks like **Figure 6**.

Now go ahead and launch Teams and @mention the GUID of your bot. This GUID is the app ID of your bot registration.

Alternatively, you can also launch Microsoft Teams by clicking on the link shown in **Figure 7**.

Type "aboutme" and you should see the bot respond as can be seen in **Figure 8**.

If you're inclined to do so, you can also package your bot as a zip file and sideload it into a Team. In the channel, you can talk to the bot directly using **getchannels** and you should see the bot respond with the list of channels in a specified Team. In my case, I had only one channel, as can be seen in **Figure 9**.

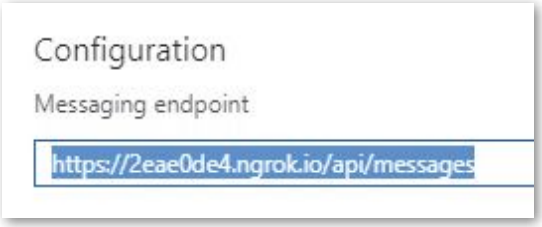


Figure 6: Update the endpoint

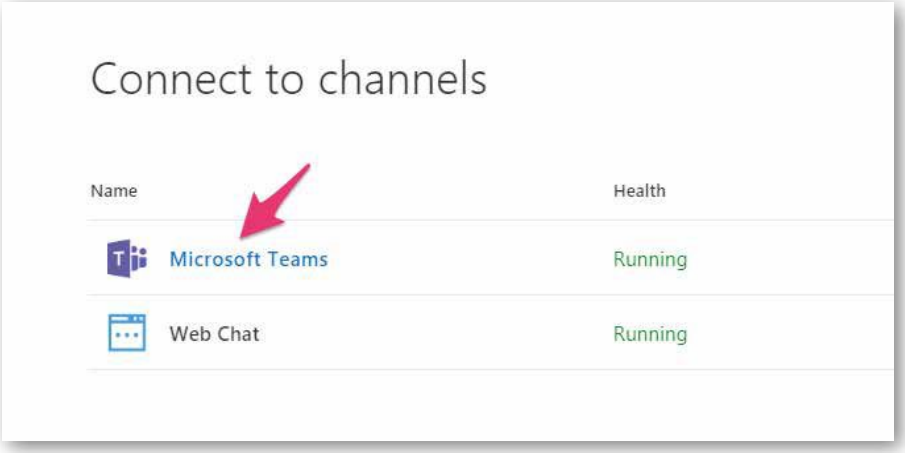


Figure 7: Launch the bot in Teams

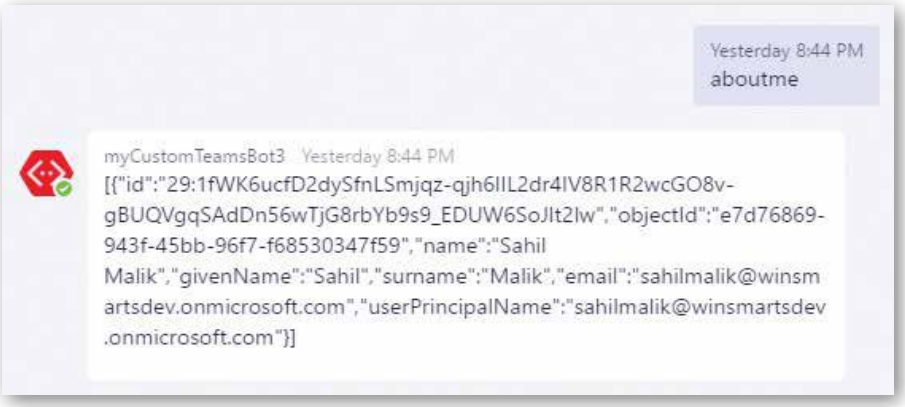


Figure 8: The bot responding to "aboutme"

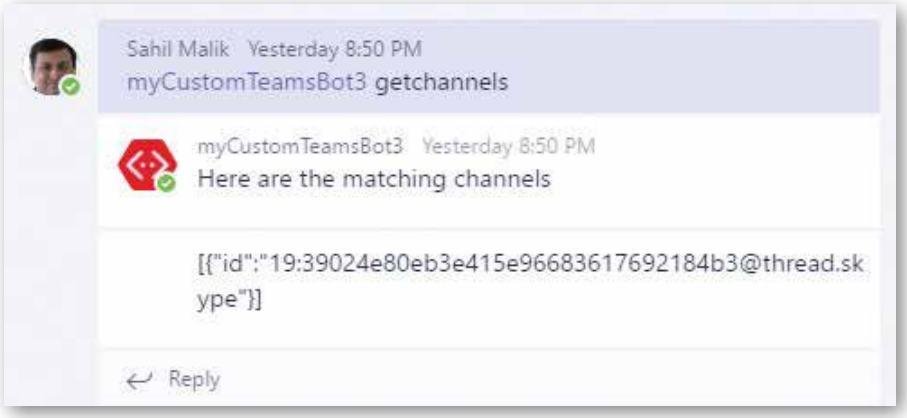


Figure 9: The list of channels, as reported by my bot

Summary

I'll share an interesting side note about myself. My education is both in computers/engineering and biology. I never wanted to be a doctor because I just couldn't get over all those long complex names, or dealing with disease and misery every day. One of my favorite topics in biology was genetics. It was quite mathematical, it involved calculating probabilities, and I always did well in that topic, while my fellow biology-inclined classmates were baffled by it. Another topic very near and dear to me neurology. There was a surprising amount of physics involved. I found it amazing that our spinal cords carry current, and that our neurons have voltage difference between the top and bottom. Incredible! I also found it amazing that when I touch a live wire, I have a reflex reaction where my brain doesn't participate in the decision making, because this is considered a life-threatening proposition. The time required for the brain to act is too long, so the neurons make a decision. The time required is about the same as an electric impulse to travel the distance of about a meter, the distance between my fingertip and my head.

Our bodies are walking and talking computers! They're capable of conversations, touch, emotion, love, hunger, and so much more.

My computer-oriented head led me to think that computers are essentially very simple brains. Note that when I was learning about neurons, the computer I had access to didn't have a hard disk and it had an 8-bit processor. But even today, computers can be considered simple. They definitely produce computation and not intelligence. But, they have the same components—thinking/processing, data collection, learning experience, just as we do. As a system becomes more intelligent, it inherently also becomes a bit more unreliable. Isn't intelligence merely a higher level of computation?

A very long time ago, I wrote an article for CODE Magazine (http://www.codemag.com/article/0807031), I'll repeat from the section under "Trim the UI" here:

"I've laid out the following items in order of complexity.

- *Light Bulb: A light bulb is rather simple. You flick a switch and 99.999% of the time, it turns on.*
- *Dog: You pet a dog and it wags its tail. Sometimes it may bite.*
- *SharePoint: More on this shortly.*
- *Men: It's not that flicking the light switch should turn the bulb on, but first we must question, is light truly what we need? Let's call a meeting.*
- *Women: Well, I'm a man, so I can't understand them myself, much less explain them to you.*

SharePoint, as you can see, is more complex than a dog, but less complex than a human. Isn't that what most computer software is becoming anyway? You flick a switch, and the bulb turns on-most of the time."

I think, we're entering a phase where computers are becoming smarter than a light bulb, maybe even smarter than your dog, but not yet as smart as humans!

What's important is that computers will be available in a range of intelligences, and it's up to us to choose how to apply them to our needs. There could be simple computers whose only job will be to reliably inform us of a water leak or an intruder. Or there could be more complex computers that could drive a car, tell a joke, paint a painting, or compose a poem. Yes, I do feel that computers will do all this one day and it will improve our lives.

We'll be like the most pampered pet of our computer overlords. It's a good thing I'm cute. I wonder what your game plan is.

For now, we're far away from that day, so let's learn the Bot Framework and a number of other interesting things

Happy coding.

Sahil Malik

CODE

Legal Notes: Potpourri

In this issue, I delve into several recent updates where legal and technology issues cross paths. DISCLAIMER: This and all Legal Notes columns should not be construed as specific legal advice. Although I'm a lawyer, I'm not your lawyer. The column presented here is for informational purposes only. Whenever you're seeking legal advice, your best course of action is to **always** seek advice from an experienced attorney licensed in your jurisdiction.

John V. Petersen
johnvpetersen@gmail.com
about.me/johnvpetersen
@johnvpetersen

John is a graduate of the Rutgers University School of Law in Camden, NJ and has been admitted to the courts of the Commonwealth of Pennsylvania and the state of New Jersey. John is a counsel with the Law Offices of Daniel M. Hanifin in West Chester, PA.
John's latest video focuses on open source licensing for developers. You can find the video here: https://www.lynda.com/Programming-Foundations-tutorials/Foundations-Programming-Open-Source-Licensing/439414-2.html.

Facebook Reverses Course and Ditches Its ReactJS Licensing Scheme

A few issues ago, I addressed the absurd licensing scheme Facebook concocted with its ReactJS JavaScript Library. When React was first released, Facebook used the BSD license plus something Facebook purported to be a patent promise. Facebook attempted to explain their intent in this blog post: https://code.facebook.com/posts/112130496157735/explaining-react-s-license/. The main problem is that although on one hand Facebook claimed that ReactJS was open source, on the other hand, the licensing scheme was not open source. The Adobe Software Foundation (ASF) called the BSD + Patent Promise a "Category X" license. As you can imagine, and as I explained in the January 2017 issue of CODE Magazine, the impedance mismatch with Facebook's licensing scheme was a fool's errand. Fast forward to September 25, 2017 when Facebook announced that ReactJS would be licensed under MIT. Between BSD and MIT, I find MIT preferable because it allows sublicensing with different licenses, whereas BSD is silent on the matter. In my opinion, it would have been better if Facebook had settled on the Apache V2 license because Apache is more explicit in its language and is both open source and commercially friendly. In any case, it's good to see that Facebook listened to the community. I don't know of anybody outside of Facebook that thought the BSD + Patent Promise was a good idea.

The New EU General Data Protection Regulation (GDPR) Goes into Effect in Just Over Six Months

Many of you who read CODE Magazine have cause to deal with the European data privacy laws where the rules and regulations are quite strict on how personal data is handled. Those rules are about to get even more strict! One of the biggest changes is the territorial coverage. Soon, if you're a European company, regardless of where you process data, you're subject to GDPR. Another interesting feature is what the GDPR calls the right to be forgotten. The GDPR gives people the right to demand that companies erase a person's data such that they not only won't be contacted any more, the company won't have physical custody of that person's data. For more information on the GDPR, follow this link: http://www.eugdpr.org/.

Copyright Enforcement: It's Becoming Big Business

To review, a copyright is the exclusive legal right given to an originator or an assignee to print, publish, perform, film, or record literary, artistic, or musical material, and to authorize others to do the same. Typically, a creator of work earns money from their work by either selling copies of the work or licensing others to do the same. The problem today for creators is that in this digital age, it's easy to simply copy and use another's work without paying for the right to do so. In some cases, this is perfectly fine when the creator declares that something is in the public domain (i.e., doesn't enforce their copyright) or uses a licensing scheme like Creative Commons. Increasingly, people who operate websites or have apps in the market place are getting demand letters from companies, often law firms that represent creators. These letters often demand huge sums of money because they claim material is being used without the creator's permission. If you get a letter like this, DON'T IGNORE IT! Just because you get a demand letter, doesn't mean you're in trouble. There are many things that a copyright plaintiff must do to both assert and prevail on a claim. The letter itself is just an assertion, not proof of copyright ownership. These letters, often referred to as troll letters, merely make bare assertions that they're entitled to their demands. If you get such a letter, consult an attorney with experience in dealing with these matters. The last thing you want to do is ignore the letter only to have a suit filed against you that you can't defend. You may also luck out when ignoring, if the letter is all bark and no bite. In my opinion, it's not a risk worth taking.

> There are many things a copyright plaintiff must do to assert and prevail on their claim. A letter is just an assertion, not proof of copyright ownership.

Professional Liability Insurance: If You're Independent You Should Have It

In my last column, I discussed whether software developers should be held to professional standards along with my caution that if you do feel that way, be careful what you ask for. Increasingly, software developers are finding themselves on the wrong end of a lawsuit where a client claims that the developer breached the contract and the client suffered damages as a result. If you're an independent developer, there are two things you should have in place before you sign a contract to deliver software. First, create an entity like an LLC. Even if you are the only member of an LLC, you get the benefit of a corporate veil that protects your personal assets. Of course, there are ways to pierce the veil. However, if you follow the rules,

it's quite difficult for one company to pierce another company's veil to get to individuals. Second, you need liability insurance. A one-million-dollar liability policy may seem like a lot, but it's probably the minimum you should carry. If you sign a contract, it's quite likely that the contact requires you to not only have insurance, but to show proof of insurance. Whether the client asks for it or not is another matter. If you get sued, you want to mitigate your risk with insurance, not your personal savings or your house!

> Copyleft is about making sure downstream code continues to have the same license terms as the original code.

Introducing the New Linux Foundation Community Data License Agreement

Taking a cue from FOSS (Free Open Source Software) licenses, the Linux foundation has created an open license that's geared toward data. FOSS licenses like BSD, MIT, Apache, etc., don't work well in the non-software context. This is why the Creative Commons licenses were established for creative works. The Linux Foundation Community Data License (CDL) comes in two flavors: Shareable and Permissive. The shareable license can be found here: https://cdla.io/sharing-1-0/. The sharable licenses embrace what are known as Copyleft rights. Copyleft is about making sure downstream code continues to have the same license terms as the original code. The permissive license can be found here: https://cdla.io/permissive-1-0/. Permissive licenses don't require downstream code to be shared. As the category name implies, permissive is about affording people the maximum amount of flexibility possible such that people can use, modify and share (or not share) as they please.

John V. Petersen

The EU General Data Protection Regulations

You can find more information about what's required and how to comply at this site: http://www.eugdpr.org/

Better Extract/Transform/Load (ETL) Practices in Data Warehousing (continued)

In the September/October 2017 issue of CODE Magazine, I wrote a Baker's Dozen (13 Productivity Tips) article on ETL Practices in Data Warehousing environments. Soon, I intend to follow that article up and provide examples of the practices discussed in the first article. Meanwhile, I want to answer some follow-up questions I've received. Some people have emailed me or approached me at speaking events where I also discuss this topic, and raised some very good points that I want to cover here.

Kevin S. Goff
kgoff@kevinsgoff.net
http://www.KevinSGoff.net
@KevinSGoff

Kevin S. Goff is a Microsoft Data Platform MVP. He has been a Microsoft MVP since 2005. He is a Database architect/developer/speaker/author, and has been writing for CODE Magazine since 2004. He is a frequent speaker at community events in the Mid-Atlantic region and also spoke regularly for the VS Live/Live 360 Conference brand from 2012 through 2015. He creates custom webcasts on SQL/BI topics on his website.

Has someone ever asked you a question and your first reaction was, "Wow, that's a great question, and I never thought to word it that way!" In the last month, two individuals asked me about identifying the mistakes that people make in building data warehouse and DW/ETL applications. For as much as I wrote about repeatable practices in my first article, I could have covered this specific question as well. I've made mistakes in my career and I've seen mistakes in other applications. In some instances, the mistakes are quite clear. In other cases, the mistake is part of a complicated situation where maybe increased awareness or foresight could have led to a better outcome. So, I'm going to cover some of the mistakes I've seen in this area.

I've been writing for CODE Magazine since 2004. I probably worked harder on the article I wrote in the September/October 2017 issue than any other article in over a decade. The subject matter of ETL in Data Warehousing is very deep, and the topic is one that is immensely personal for me as well. Borrowing from Meg Ryan's character in the movie "You've Got Mail," I've always thought that making things personal isn't necessarily a bad idea: That's where passion comes from, and some things ought to **begin** by being personal!

Having said that, after reading the article again, I realize that there were some points I want to amplify, and a few points I neglected to raise. So, I'll take some time and cover these points.

After I wrote this latest article, I asked myself the question that every writer should always ask: Who is your intended audience for this article? That reminded me of a story. (Anyone who knows me knows I'll always jump at the chance to use some random piece of trivia.)

This is a story about the great baseball player, Willie Mays. One night, his long-time manager Leo Durocher asked him a series of questions related to in-game strategies: "It's the bottom of the ninth and the score is tied. We've got a runner on third and one out. The current batter is in an 0-20 hitting slump, and the pitcher is this-and-this-and-that. What should the manager do?" Mays quickly replied, "I'm a ballplayer, that's the manager's job." Durocher responded that Mays needed to appreciate these situations to gain a better understanding of the game, that it would ultimately make him a complete player. Later Mays credited Durocher with helping him to see that point.

So why am I mentioning this? To this day, there are debates about heads-down programmers versus developer/analysts, waterfall methodologies versus the Agile/Sprint world, etc. Willie Mays' initial response to his manager might seem like a perfectly appropriate response for a heads-down programmer who reads this article. (Before I continue, I don't use the term "heads-down" programmer in any disparaging way. I'll get to that shortly).

A programmer who prefers to work from specs might read this article and respond, "OK, there's some interesting stuff here, but honestly this sounds like it's more for managers and tech leads, and I'm a programmer".

> Every aspect of data warehousing is about context.

Fair enough: At different stages of my career, I wanted to just write code and build apps and let others work with the business side. But I came to realize that this kind of insularity just isn't practical when developing data warehouse and business intelligence applications. Perhaps in certain areas of IT that are focused on scientific/engineering/systems software, being a heads-down programmer is perfectly fine, but in the world of business applications, it's not. Every aspect of data warehousing is about **context**. Programmers will admit that even the best and most complete specification can't include every major circumstance, every factor that goes into some aspect of the data and how users will work with it. A programmer needs to understand all the nuances of source data. A programmer needs to understand that various business people don't necessarily have the same understanding of a piece of data. A programmer needs to understand both the system and user dependencies.

In other words, a programmer needs to at least appreciate what a team lead or manager is responsible for.

Remember, the average turnover rate in this industry is less than two years. You might think that you're "just a ballplayer" this year, but next year you might be thrust into the role of a team lead or player-coach. The more versatility you can show, the more valuable you'll be.

Some technical programmers who work from specifications might not initially feel they have a compelling need to understand this information today. I'd argue that I once felt that way, but came to realize that I was missing a big part of the picture. Everyone on the data warehouse team, from the programmers to tech leads to architects to QA managers, needs to understand the big picture.

What's on the Menu?

Here are the six topics for today:

1. Common mistakes in ETL applications
2. Operational Data Store (ODS) Databases
3. Capacity Planning
4. Using Change Data Capture and Temporal tables in Data Warehousing
5. Data Lineage documentation
6. Discussion topic: Do you need technical skills to identify business requirements?

#1: Common Mistakes in Data Warehousing and ETL Applications

If you search the Web for "Why Data Warehousing Projects fail", you'll see many online articles and blog posts with stories about failed projects. Projects can fail for many reasons, technical and non-technical, strategic and tactical, etc.

There's enough information available to fill a book on this topic, so I won't get into all the reasons a project can fail. For that matter, a project can succeed in the sense that the team meets core deadlines and delivers base functionality, but the application might still fall well short of an optimal outcome. Here are some of the mistakes that tend to occur in Data Warehousing and DW/ETL projects, in no order of importance:

- Insufficient Logging
- Insufficient time spent vetting and testing the incremental extract process
- Insufficient time spent profiling the source data
- Not having a full understanding of the transactional grain (level of detail) of all source data
- Not taking enough advantage of newer technologies
- Less than optimal SQL code in the ETL layer
- Uneasiness/challenges in getting the business to agree on rules
- Not retaining historical data
- Building a solution that works but doesn't get used
- Not using the Kimball model (or any model) and not using accepted patterns
- Not running ideas by team members

Let's examine each of these points:

Insufficient Logging: In my previous article, I stressed the importance of logging all activity in the ETL application. This includes capturing information on extracts from the data source, loading into the staging area, and populating the eventual data warehouse model. The team needs to think about everything they might want to know about an overnight ETL job that occurred three nights ago. When you design logging processes, you're essentially writing the context for your history. Just like viewers of the TV show "The Brady Bunch" are accustomed to hearing, "Marcia, Marcia, Marcia!", people on my data warehousing team have come to learn my corollary, "Log, Log, Log!"

Insufficient time spent vetting and testing the incremental extract process: If extracts from the source systems aren't accurate or reliable, the contents of the data warehouse are going to be incorrect in some way. Even a few missed invoices or cost records because of some data anomaly or unexpected issue can impact user/business trust of the data warehouse.

There's a corollary here as well: **Insufficient time was spent profiling the source data.** The degree to which you know the source data, including source system behavior, how it handles timestamps, etc., will likely be the degree to which you can design an effective and reliable extract strategy. There's the old line about police officers needing to be right every time, and criminals only needing to be right one time. The same thing holds true for data warehouse incremental extracts from source systems: You need to be right every time.

Not having a full understanding of the transactional grain (level of detail) of all source data: I can't stress enough that if you don't have a firm understanding of the grain and cardinality of the source data, you'll never be able to confidently design extract logic.

Not taking enough advantage of newer technologies: I've known very solid developers who write bug-free code, and yet their applications don't take advantage of newer features in SQL Server for performance. This is more of an issue for architecture leads who need to evaluate newer database enhancements to determine whether the team can leverage them. As one example, the optimized In-Memory OLTP Table capability (first introduced in SQL 2014 and greatly enhanced in 2016) can potentially increase staging table processing by a factor of two to three times, or greater.

True, a slow but reliable application is far better than a fast one with flaws. Additionally, we all know that implementing newer functionality for performance carries risks that need to be tested. Still, it helps to pay close attention to what Microsoft adds to the database engine.

Less than optimal SQL code in the ETL layer: I see this one often, and admittedly I fall victim to this, too, if I'm not careful. Because many DW/ETL operations occur overnight and can take hours, there's a tendency to adopt a "well, so long as the whole thing runs between midnight and 4 AM, we're good" mindset. To this day, I see people using SQL CURSOR logic to scan over a large number of rows, when some refactoring to use subqueries instead and maybe some ranking functions would simplify the code and enhance performance. Never hesitate to do code/design reviews of SQL queries in your ETL!

The "Other" DDL: Documenting Data Lineage

Documenting data lineage can be a tedious and painful task. But there's something even more painful: Trying to figure out where data comes from. Even though we're all developers, the answer of "I'll have to look it up in the code" gets old. Recently, I had to document the lineage of about 200 data elements. I made a silly game out of it. Whatever gets it done, right? My team now has a document that we've been wanting for months.

Uneasiness/challenges in getting the business to agree on rules: I've seen good developers and even team leads who are uneasy about approaching either the business side or even the technical leaders of a legacy source system when they see inconsistencies in business requirements. Here's the bottom line: This is not the time to be shy. If necessary, vet your questions among the team members to make sure you're asking a solid question with good foundation.

There's the story about a team who discovered that two different segments of business users had been using two different definitions for what constituted capital finished goods, based on different process and disposition rules. The DW team was apprehensive about approaching either team to discuss differences/discrepancies. Should the organization follow one uniform rule for finished material, or could the organization carry forward two rules with distinct names and contexts? You'll never know until you raise the issue. Believe me, if you fail to raise the issue, you shouldn't expect someone else to raise it. Once again, this is not the time to be shy. As someone once observed about Hugh Laurie's character in the TV show House, "House thinks you're being timid—that's the worst thing to be, in his book."

Not retaining historical data: Here's a cautionary tale that every data warehouse person should remember. Years ago, a company wanted to do a complete refresh of data from the source systems. Unfortunately, the source system had archived their data, and as it turned out, some of the backup archives weren't available. (Yes, the situation was a mess). Had the ETL processes retained copies of what they'd extracted along the way before transforming any of the data in the staging area, the data warehouse team might not have needed to do a complete refresh from the source systems. If you're a DBA, you're probably feeling your blood pressure rising over the thought of storage requirements. Yes, good data warehouses sometimes need high storage capacity.

> Agile methodologies, Sprint processes, and frequent meaningful prototypes are tools and approaches. By themselves, they don't guarantee success.

Building a solution that works but doesn't get used: OK, we've all built an application that wound up sitting on the shelf. It happens. But when it happens in the same company or with the same team multiple times, you need to look for reasons. My teams prototype and work directly with users every step of the way. I make sure that no more than two weeks pass without some meaningful demo of where we are with respect to getting new data into the data warehouse. The demos reflect, as much as possible, how we believe users will ultimately access the application. We seek sanity checks constantly. At times, I probably drive the business people crazy with communications. There's never a guarantee that the business will use the system in the intended fashion: but managing the development of a project is about taking the right incremental steps along the way. Having said that, Agile methodologies, Sprint processes, and frequent and meaningful prototypes are only tools and approaches. By themselves, they don't guarantee success.

In short, when users don't use the final product, quite likely there were serious communication gaps along the way.

Not using the Kimball model (or any model) and not using accepted patterns: I've seen failed or flawed implementations of fact/dimensional models and ETL modules written by some (otherwise) very skilled database developers who would have fared much better had they been carrying some fundamentals in their muscle memory. Always remember the fundamentals!

Not running ideas by team members: Even with demanding project schedules, I set aside a few hours on Friday afternoons for my team to share ideas, talk through issues, etc. It's an open forum. During our morning scrum meetings, if someone has a technical or design issue they want to talk about, we table it for Friday afternoon. I'm fortunate that I have very good team members with whom I look forward to collaborating. Team chemistry is a complicated topic and I certainly don't claim to have all the answers, but I've learned over time that you have to bring your "best self" into these meetings. When a data warehouse is only understood by a subset of the team members because other members weren't involved in talking through specifics, ultimately some aspect suffers.

#2: Operational Data Store (ODS) Databases

In my prior DW/ETL article (September/October 2017 CODE Magazine), I showed a figure similar to **Figure 1**: a very high-level topology/roadmap for a Data Warehouse architecture. I've updated **Figure 1** from that article to include an optional but sometimes key component: an operational data store.

I've used the joke many times about 10 economists presenting 10 different economic models, where it can be difficult to tell whose model makes the most sense. In the same manner, you could present technical requirements to 10 data warehouse professionals and easily get the same number of approaches.

When I presented **Figure 1** in the previous article, I talked about ETL processes reading data from the source systems into a staging area, and then from the staging area into the data warehouse. Well, sometimes a company might introduce what's called an operational data store (ODS) into the picture, either in addition to the data warehouse or, in some cases, in lieu of the data warehouse. You can think of an ODS as an intermediary, or "middle-tier" storage area: a stepping-stone along the way toward a data warehouse. There are several reasons a company might do this, and here are two that I've experienced first-hand:

- You might implement an ODS if the source system offers little or no transactional/operational reporting. As a result, you might implement transactional daily "flash" reporting out of the ODS, and then more comprehensive analytic queries from the data warehouse.

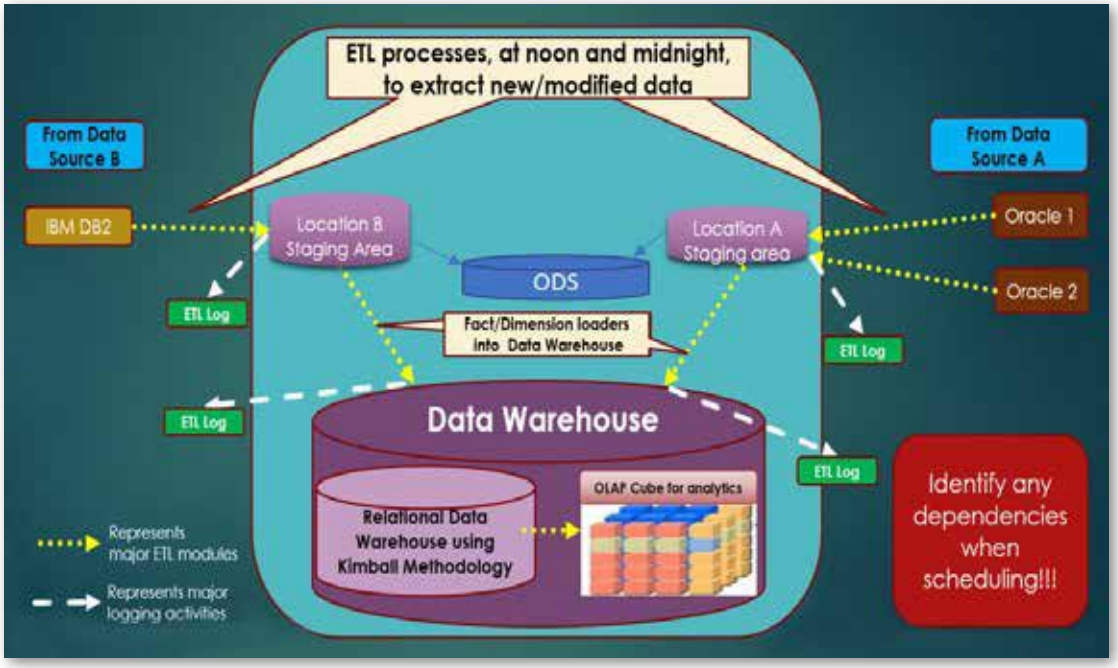


Figure 1: An updated high-level topology picture of an ETL architecture

- You might implement an ODS if you have multiple staging areas for different source systems, and want to use a single common intermediary area as the source for the data warehouse. Additionally, an ODS can simplify verification queries when the staging areas contain many complex tables.

#3: Capacity Planning

I have an opinion on Capacity Planning (a document for anticipated system growth over three-to-five years and projected hardware/storage costs): No one ever accurately predicts the actual capacity in advance. It's a matter of whether you're in the ballpark or out in the next county.

At the end of 2017, I updated a capacity planning document I built in 2015. My original numbers were off, although not horribly so, and I was generally able to defend what I accounted for two years earlier. Having said that, there were still lessons learned. So, when determining the anticipated growth of your databases, factor in these four items:

- Any new or anticipated subject matter data, data feeds, etc.
- Backups
- Historical/periodic snapshots
- Redundant information in staging areas and operational/intermediary areas

First, account for any new or anticipated subject matter data, data feeds, etc. Granted, if 12 months from now the business asks you to incorporate a data feed that no one's discussing today, you can't account for what you don't know. Having said that, try to know as much as you can. Try to find out if company executives have any interest in seeing new data.

Second, take backups into account. Take a measure of how many backups you want to retain on your server (for easy restoration), and your backup policy in general.

Third, take historical snapshots into account. If you're taking complete or partial snapshots every month (for accounting/reporting purposes, Sarbanes-Oxley requirements, etc.), remember that those snapshots take up space.

Finally, make sure to factor in redundant information in staging areas and operational/intermediary areas. Remember that storing legacy data in one of your data stores adds to your physical storage requirements.

I maintain a spreadsheet of all items to account for, and I've learned to periodically review the contents (along with database growth patterns) with my team. One suggestion is to have a quarterly meeting to review capacity planning.

#4: Change Data Capture and Temporal Tables and Their Place in the Data Warehouse Universe

At some point in the Data Warehouse process, either reading from source systems or loading into the data warehouse or somewhere in between, developers want to capture "what got changed and what got added." Sadly, developers also have to deal with "what got deleted," even though we all know that the best systems flag transactional rows for deletion (or send reversing entries) but don't physically remove them. <smile>

Certainly, capturing what was inserted and updated isn't unique to data warehousing: Many applications need to store audit trail logs and version history so that you can see who added/changed data and when.

Until SQL Server 2008, the only way you could capture changes in SQL Server was to do one of the following:

- Implement database triggers to capture inserted/updated/deleted data.

Know the Source Data as Well as the OLTP Developers Who Manage It

Many things can negatively impact the perception of a data warehouse application. One of them is when developers of the source systems perceive that the DW team has a superficial understanding of the source data. Fair or unfair, the best DW/ETL developers and architects need to understand the data they're extracting as well as the source system developers who maintain it. Once a legacy manager (rightly) ridiculed me for not knowing enough about their transaction system. Then I buckled down and studied it to the point where I found things he wasn't aware of. He changed his tune.

- Implement logic in stored procedures (or in the application layer) to capture inserted/updated/deleted values.
- Purchase a third-party tool, such as SQLAudit.

All of them worked, although each solution harbored challenges to keep an eye on. Database triggers are very reliable, but they're a very code-intensive solution and can introduce performance issues if you're not careful. Implementing logic in stored procedures or the application layer can also work, even though it means that every module writing to the database must adhere to the logic. Third-party tools can provide excellent functionality, yet you can encounter licensing challenges and possibly other challenges from using a solution that isn't as configurable as you'd like.

Some criticized Microsoft for not providing a built-in solution in the database engine to capture and store changes. Those criticisms reflected that in SQL Server 2005, the Microsoft database platform was OK for medium-sized databases, but it lacked the core functionality needed in data warehousing circles. Even though I've loved SQL Server from day one, those critics had a valid point.

Fortunately, Microsoft responded with functionality in SQL Server 2008 called Change Data Capture (CDC). For years, Microsoft only made CDC available in the Enterprise Edition of SQL Server, but relaxed that restriction and included CDC in Service Pack 1 of SQL Server 2016 Standard Edition. CDC monitors the transaction log for database DML operations, and writes out a full log of the before/after database values to history tables that you can query. Because it scans the transaction log asynchronously, you don't need to worry about additional processing time for the original DML statement to execute. CDC is an outstanding piece of functionality that provides database environments with a reliable way of capturing changes with very little code.

If you want to learn more about CDC, I wrote about it back in the September/October 2016 issue of CODE Magazine. Here is the link to the article: (specifically, look at Tips 3 and 4) http://www.codemag.com/Article/1009081/The-Baker's-Dozen-13-More-Examples-of-Functionality-in-SQL-Server-2008-Integration-Services.

Also, I created a webcast on CDC back in January of 2014. If you navigate to my website (www.KevinSGoff.net), and go to the Webcast category on the right, you'll see a list of webcasts, including the one on CDC.

In 2016, Microsoft added a second piece of functionality called temporal tables. At first glance, temporal tables seem like "CDC-light." Temporal tables collect changes to data and write out version history to a history table. I wrote about temporal tables in the September/October 2016 issue of CODE Magazine (specifically, Tip #9): http://www.codemag.com/Article/1609061/The-Baker's-Dozen-13-Great-Things-to-Know-About-SQL-Server-2016.

I'm not going to promote one feature over another, as each has its place. A good data warehouse architect should be aware of these features, what they provide, and what they don't provide (at least out-of-the-box). So here are six things to keep in mind.

First, if the source system is a database system with no audit trail history and no way to capture changes, all you can do to extract data from a source system is to have a reliable means of detecting inserts/updates based on the management of the last update/timestamp values in the source data.

Second, source systems might have their own version/implementation of CDC. CDC, in itself, is not a Microsoft-specific invention; other databases, such as DB2, have had CDC for years. The question is whether the source system uses it, and whether there are any rules that they've already defined for it. Once again, ETL developers need to know as much as possible about that source system.

Third, you can use CDC in your middle layer if you're persisting transactional history in your staging or ODS layer. For instance, you might extract 1,000 invoices from the source system and merge them into your staging area. Maybe 100 invoices are new (based on the invoice key), 200 are existing invoices where some non-key field has changed, and 700 invoices are ones you already have in your intermediate layer where no values changed. You can use CDC to capture the 100 new invoices (and the old/new values of the 200 updated invoices), and send just those values to the data warehouse model (or whatever the next phase is in the process).

Fourth, as a follow-up on that last point, even if you're not using CDC, you could use the T-SQL MERGE to insert/update those 100 new and 200 changed rows (and ignore the 700 that didn't change), and OUTPUT the information to some destination yourself. Having said that, you'd have to implement/maintain the logic yourself that checks all non-key fields for changes. Some shops might build an engine that generates MERGE statements using metadata to automate this.

Fifth, some developers who evaluate SQL Server 2016 wonder if the new temporal tables feature (and support for versioning history) in SQL Server 2016 can be used in place of CDC. It's important to understand how temporal tables differ. Suppose I insert a row into a base table where I've configured temporal tables. The version history of that table is empty, because temporal tables only deal with changes. Now suppose I update that row and change a column. The temporal table stores the old version of the row along with time-stamped versions of the start/end date of the life of that row. But you'd need to examine the version history row and the base table to see which column(s) changed. Maybe you care about that, maybe you don't. However, there's a catch. Suppose I update the row in the base table, but I don't change any values. SQL Server still writes out a new row to the temporal table version history, even though nothing changed. (By contrast, CDC is intelligent enough to detect when an update doesn't change any values, and doesn't write out any change history). I consider that a shortcoming in temporal tables.

Finally, there's a shortcoming in CDC. If you perform a DDL change on a table (e.g., add a new column), CDC

Everything Is a Journey, So Keep Moving

From 2005 to 2008, I transitioned from Crystal Reports to SSRS. In 2017, I lost 55 pounds. Over the course of 15 years, I've studied the Kimball methodology. Everything is a journey, where accomplishments occur a little at a time. Sure, you can think of them as marathons, but you still need to keep moving. When a team member accomplishes anything, I tell the person, "Good…keep at it… keep the motor running."

doesn't automatically cascade that change into the change tracking history for the base table. You have to perform a few extra steps. The steps can be automated, and they certainly aren't deal-beakers, but you need to be aware that CDC doesn't handle DDL changes automatically to begin with.

In summary, I like both features. I like the intelligence that Microsoft baked into the CDC engine and I like the level of detail that CDC stores in the change tracking history tables. However, because CDC writes out just column changes, trying to reconstruct full row version history isn't a trivial task. By contrast, I like the row version history in temporal tables, but I was disappointed that the engine doesn't even give me the option of suppressing row history if an UPDATE truly didn't change column values.

Whether you include either feature in your arsenal is up to you, but it's very important to understand how these tools work and whether you can effectively leverage them in your ETL operations.

#5: Data Lineage Documentation

One of my pet peeves is seeing data teams construct documentation quickly, where the documentation has limited (or no) practical value. My conscience just doesn't permit me to quickly go through the motions and produce something for the sake of saying I produced documentation, when I know I wouldn't find it helpful if I were new to the situation.

There are individuals who acknowledge that much of the data documentation out there isn't fantastic, but also acknowledge the following:

- Producing good documentation can be very time-consuming.
- Producing good (and up-to-date) documentation can be especially difficult if the team constantly faces change/enhancement requests.
- Different team members can have reasonable disagreements about what constitutes good and useful documentation.

Although I love the Microsoft Database platform, I'll freely acknowledge that the SQL Server environment isn't inherently conducive to good data documentation. The tools certainly don't stand in the way of building good meta-data, but they don't exactly offer what I consider to be good tools to facilitate it. So, the less-than-sanitized little secret is that if you want to build good data documentation that other developers and maybe even some analysts will find useful, you have to build it yourself.

Although I've summarized here for brevity, here's an example of a document (**Figure 2**) that I recently worked with my team to build. For every data element, we define the system data type and how we treat the element (was it system generated, a business key, a straight feed from the source, or a derived column that's based on lookup rules or other rules). The source lineage is in narrative form and sometimes contains several lines of CASE statements. At the end of the day, you base the format on what works for you.

#6: Just How Important are Technical Skills When Defining Requirements?

Recently, at a SQL Saturday event, I engaged in a spirited but friendly lunchroom debate about the role of technical skills in identifying business requirements. Although I've never conducted a formal poll, I'll bet the price of a fancy dinner that more people would say that those creating good solid business requirements should not need technical skills. Well, I'm going to go against the predominant view in our industry and state boldly that technical skills are often necessary to the process of defining clear business requirements for a Data Warehouse application.

> In some cases, I've been able to "cut to the chase" of hashing out business requirements by building prototypes and showing them to users. Without this approach, we might never have identified some of the more intricate details.

We could have a separate debate about whether it should be necessary in the ideal world, but we live in a world that's often messy. So here are three compelling reasons why I say tech skills are realistically important in hammering out requirements.

First, often I'm able to work through specific user requirements by building prototypes and meaningful demos.

DimCustomerOrder documentation

Data Element	Data Type	Type of Attribute	Source Lineage
CustomerOrderPK	Int identity	System-Generated PK	
CustomerOrderNumber	Varchar(30)	Business Key	Assigned by OLTP order entry
OrderProgramName	Varchar(30)	Generated Attribute	From lookup table OrderProgramLookup, based on OLTP field OrderProgramCode
PartWidthRange	Varchar(30)	Generated attribute	From lookup table WidthRange, based on OLTP field PartWidth

Figure 2: A short except from a sample data lineage document

Business users are often very busy and don't necessarily have the time to read/build meaningful specs. They need to see something live and "shoot down" the wrong assumptions in the demo to get to what they're really after. Even if you work in a highly-regulated industry such as health care, even within the rules, there can be all sorts of areas of variation that won't get covered in a vacuum. Obviously, it's going to take someone with some technical knowledge to build a good demo. So yes, sometimes you need to use the technology as a means to gathering user requirements.

Second, I've seen cases where business users have lulled themselves into a false sense of security about legacy systems. The business might prepare what they believe to be fully detailed and bullet-proof specifications on what they want in a new data warehouse/analytic system. But their false assumptions might lead to a "garbage in, garbage out" scenario. The final data warehouse solution might work according to specification, but might only serve to propagate an undesired aspect into a new system. This could be anything, including the business having conflicting views about the nature of historical data, or a misunderstanding about business logic. Remember, businesses deal with turnover just like IT does—they patch their rules and their understanding over time, just like legacy systems get patched. On more than one occasion, I've spotted serious issues in source data long after the business wrote the requirements. Had a technical person participated in the requirements and profiled the data properly, developers wouldn't have gone down the wrong path.

Finally, a person with good technical skills can sometimes review requirements and improve them. When I was a rookie developer, my boss (a developer manager) sat in on a business meeting with a project manager who was gathering requirements. My boss had a fantastic idea in this phase that immediately generated great interest with a business team that was previously lukewarm on the new system. My boss was one of those types who could see the entire picture: the technology, the use, the implementation, everything. Data Warehouse projects always need the big picture people, and the big picture includes technical skills.

To my colleagues in the database world, your statements that a person can build good data warehouse specs and requirements without tech skills are well-intentioned. Sadly, reality says otherwise.

Patience and Balance: A Periodic Reminder to Actively Study the Kimball Methodology

Once a month I try to reread a chapter or at least part of a chapter in the Kimball data warehousing/dimensional modeling books. I set aside time to read the design tips on their website. I've read them many times, but to this day, I still catch something that didn't fully register before. Other times, I'll have forgotten some scenario, only to have some information jog my memory.

When our children complain about getting up to go to school every morning, we try to find the words to convey the concept of a lifetime of learning. We need to carry that passion into our own work, but we also need to be patient. Sometimes it's difficult to keep even one toe in the nice tranquil waters of learning while the other 99% of our body feels the heat from the scalding waters of deadlines, difficult projects, etc.

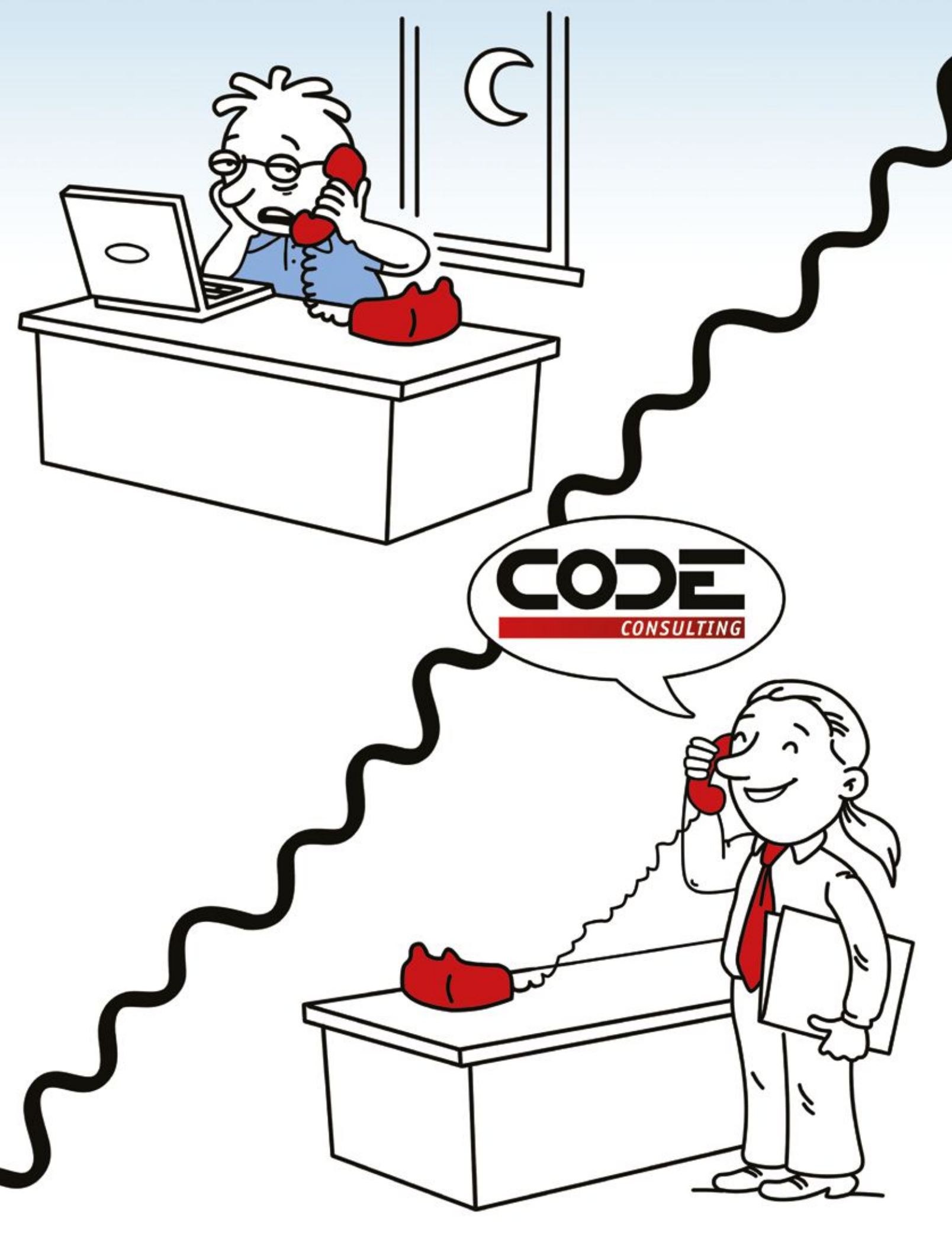
It takes time to go through a learning curve and even longer to incorporate what you've learned into active projects. For some, it's months, for others, it can be years. Perhaps the only thing worse than having no time to learn new things is actually learning things and being frustrated that you can't apply any of it because your current projects won't permit it. I know this is the easiest thing in the world to say, but it takes time to establish good practices. Be patient but keep moving. These changes don't occur overnight: They're organic and seminal. The bridge between a dream and success is hard work, and you must pay close enough attention to seize any opportunity to more forward!

> It bears repeating:
> Good practices are organic and seminal. It takes time to achieve repeatable success.

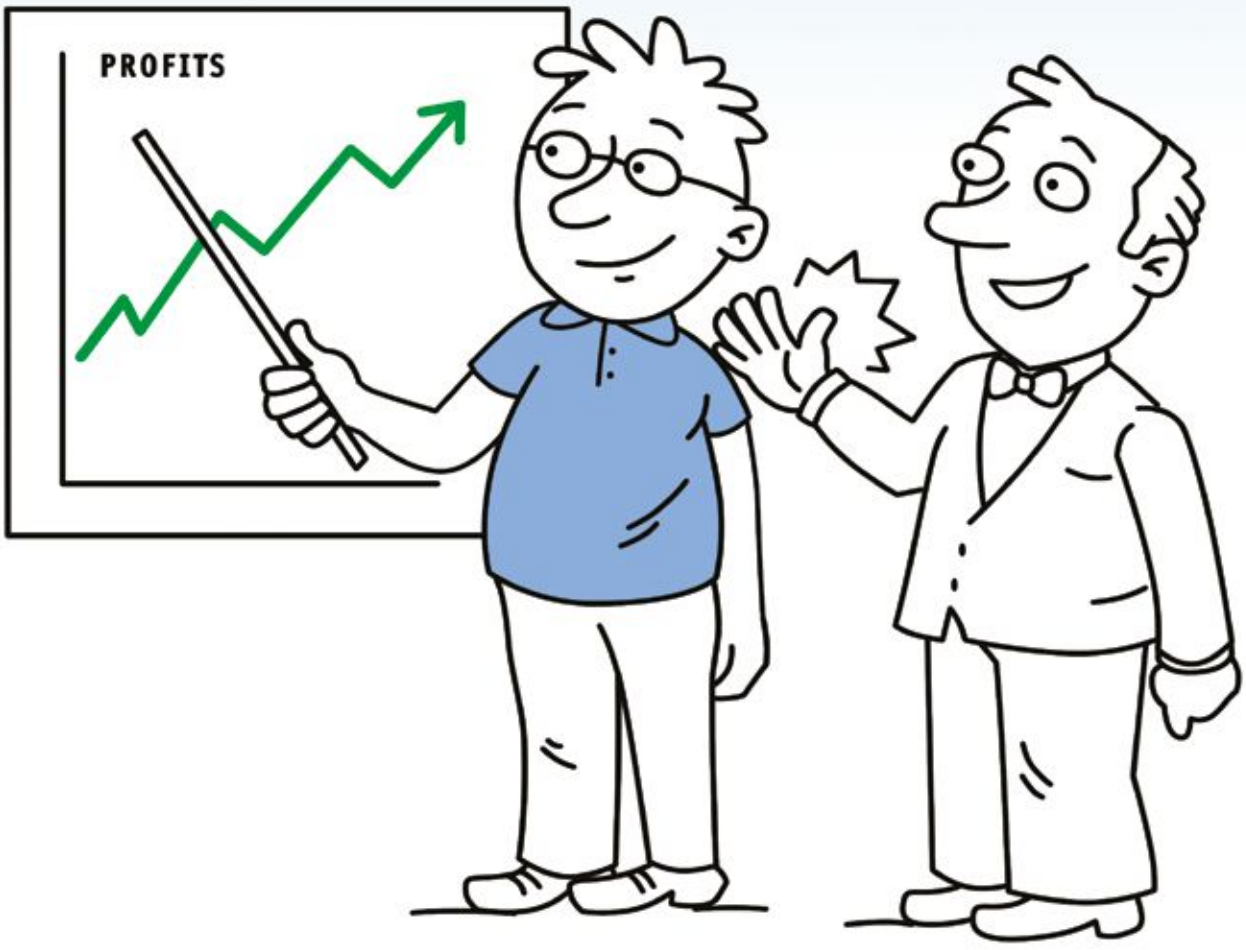
Tune in Next Time…

Many years ago, I worked on a large public-sector application for which I won awards from the United States Department of Agriculture. I've been refactoring it to create a community data warehouse ETL demo. In my next article, I'm going to present portions of the ETL application and will reference the content from this article. Stay tuned!

Kevin S. Goff


CODE
CONSULTING

CODE Consulting Will Make You a Hero!



Lacking technical knowledge or the resources to keep your development project moving forward? Pulling too many all-nighters to meet impending deadlines? CODE Consulting has top-tier programmers available to fill in the technical and manpower gaps to make your team successful! With in-depth experience in .NET, Web, Azure, Mobile 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.

Helping Companies Build Better Software Since 1993

www.codemag.com/techhelp
832-717-4445 ext. 9 ● info@codemag.com

CODE CONSULTING

Ready for Prime Time: .NET Core 2.0 and ASP.NET Core 2.0

Many of us have been patiently waiting through the long and winding road that has been the inception of the .NET Core and ASP.NET Core platforms. After a very rocky set of 1.x releases, version 2.0 of the new .NET Frameworks and tooling finally arrived a few months back. You know the saying: "Don't use any 1.x product from Microsoft," and this is probably truer than ever with

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.

.NET Core and ASP.NET Core. The initial releases, although technically functional and powerful, were largely underfeatured and sported very inconsistent and ever-changing tooling. Using the 1.x (and pre-release) versions involved quite a bit of pain and struggle just to keep up with all the developments along the way.

.NET Core 2.0 and ASP.NET Core 2.0: The Promised Land?

Version 2.0 of .NET Standard, .NET Core, and ASP.NET Core **improve the situation considerably** by significantly refactoring the core feature set of .NET Core, without compromising all of the major improvements that the new framework brought in version 1.

The brunt of the changes involved bringing back APIs that existed in the full .NET Framework and make .NET Core 2.0 and .NET Standard 2.0 more backward compatible. It's now vastly easier to move existing full-framework code to .NET Core/Standard 2.0. It's hard to understate how important that step is, as 1.x was hobbled by missing APIs and sub-features that made it difficult to move existing full-framework code and libraries forward to .NET Core/Standard. Bringing the API breadth back to close compatibility with the full framework resets expectations of what's considered a functional set of .NET features, the kind that most of us have come to expect from .NET as a platform.

These subtle but significant improvements make the overall experience of the 2.0 .NET Core and ASP.NET Core releases much more approachable, especially when coming from previous versions of .NET. More importantly, third parties can now more easily move their components to .NET Core so that the support eco-system for .NET Core applications doesn't feel like a backwater, as it did during the v1 days.

These changes are as welcome as they were necessary and my experience with the 2.0 wave of tools so far has been very positive. I've been able to move two of my most popular libraries to .NET Core 2.0 with relatively little effort, something that would have been unthinkable with the v1 versions. I've also finally started building a number of internal Web applications with ASP.NET Core 2.0 and the overall feature breadth is pretty close to full framework, minus the obvious Windows-specific feature set.

ASP.NET Core 2.0 also has many welcome improvements, including simplified configuration that provides sensible defaults, so that you don't have to write the same obvious startup code over and over. There are also many new small enhancements as well as a major new feature in the form of RazorPages that bring controller-less Razor pages to ASP.NET Core.

Overall, 2.0 is a massive upgrade in functionality, bringing back features that realistically should have been there from the very beginning.

But it's not all unicorns and rainbows. There are still many issues that need to be addressed moving forward. First and foremost is that the new SDK-style project tooling is still struggling with many small missing features and performance problems. Visual Studio, in particular, seems to have taken a big step backward in stability in the .NET Core release cycles, but the good news is that Microsoft is finally taking tooling performance and stability seriously. The last few Visual Studio point release updates, even since the release of .NET Core/ASP.NET Core 2.0, have made big strides in addressing stability and performance. But there's still some ways to go.

The Good Outweighs the Bad

Overall, the improvements in this 2.0 release vastly outweigh the relatively few—if not insignificant—problems, that still need to be addressed. I've been waiting for a long time for a .NET Core/Standard version that looks production-ready. Here's where I stand: The release of .NET Core 2.0 and ASP.NET 2.0 is my demarcation line, my line in the sand, where I get my butt off my hands and finally jump in. It's been a long wait to get to this point.

> The release of .NET Core 2.0 and ASP.NET 2.0 is my demarcation line, my line in the sand, where I get my butt off my hands and finally jump in.

The 2.0 release feels like a good jumping-in point, to dig in and start building real applications. This is a feeling that I never had with the v1 releases. In v1, I dabbled and worked through my samples and learned, but with V1 releases, I never felt like I'd get through a project without getting stuck on some missing piece of kit.

For me, v2.0 strikes the right balance of new features, performance, and platform options that I actually want to use, without giving up many of the conveniences that earlier versions of .NET offered. The 2.0 features no longer feel like a compromise between the old and new but

a way forward to new features and functionality that's really useful and easy to work with in ways that you would expect on the .NET platform. Plus, there are many hot and shiny new features in ASP.NET Core. There's lots to like and always has been in ASP.NET Core.

Let's take a look at some of the big improvements in .NET Core 2.0 and ASP.NET Core 2.0.

Bringing Back Many .NET APIs

The first and most significant improvement in the 2.0 releases is that .NET Standard and .NET Core 2.0 bring back many of the .NET Framework APIs we've been using since the beginning of .NET in the full framework, but that weren't supported initially by .NET Standard and .NET Core v1.

The terms .NET Core and .NET Standard will come up a few times in this article and the concepts of the framework and specification are somewhat important. If you're new to these terms, please see the sidebar .NET Core and .NET Standard.

> .NET Standard 2.0 is a specification, a blueprint of features that have to be implemented by a specific platform. .NET Core 2.0 is a specific platform implementation of that specification.

.NET Core 1 Was Too Lean

When .NET Core 1.x came out, it was largely touted as a trimmed down, high-performance version of .NET. As part of that effort, there was a lot of focus on *trimming the fat* and providing only core APIs in .NET Core and .NET Standard. The bulk of the .NET Base Class Library was also broken up into a huge number of small hyper-focused packages.

All of this resulted in a much smaller framework, but unfortunately also brought a few problems:

- Major incompatibilities with classic .NET Framework code (it was hard to port code)
- A huge clutter of hard-to-discover NuGet Packages in projects
- Many usability issues when trying to perform common tasks
- A lot of mental overhead trying to combine all of the right pieces into a whole

With .NET Core 1, many common NET Framework APIs were either not available or they were buried under different API interfaces that were often missing critical functionality. Not only was it hard to find stuff that was under previously well-known APIs, but a lot of functionality that was taken for granted (Reflection, Data APIs, and XML, for example) was refactored down to near un-usability.

Bringing Back Full Framework Features

.NET Core 2.0—and more importantly **.NET Standard 2.0**—adds back a ton of functionality that was previously cut from .NET Core/Standard. In v1 it was really difficult to port existing code. The feature footprint with .NET Standard 2.0 and .NET Core 2.0 is drastically improved (~150% of new APIs added over v1.1) and compatibility with existing full framework functionality is preserved for a much larger percentage of code. In real terms, this means that it's now much easier to port existing full framework code to .NET Standard or .NET Core and have it run with no or minor changes.

Case in point: I took one of my **15-year-old** general purpose libraries, Westwind.Utilities, which contains a ton of varied utility classes that touch a wide variety of .NET features, and re-targeted it to .NET Standard. More than 95% of the library migrated without changes and only a few small features needed some tweaks (encryption, database). A few features had to be cut out (low-level AppDomain management and System.Drawing features). Given that this library was such an unusual hodgepodge of features, more single-focused libraries will fare much better for conversions. If you're not using a few of the APIs that have been cut or only minimally implemented, chances are that porting to .NET Standard will require few or even no changes.

You can read a lot more detail about what's involved in porting a full framework .NET library to .NET Standard in this blog post: Multi-Targeting and Porting a .NET Library to .NET Core 2.0 (https://goo.gl/e1ADWe).

Runtimes Are Back

One of the key bullet points that Microsoft touted with .NET Core is that you can run **side by side** installations of .NET Core. You can build an application and ship all the runtime files and dependencies in a local folder, including all the .NET dependencies as part of your application. The benefit of this is that you can much more easily run applications that require different versions of .NET on the same computer. No more trying to sync up and potentially break applications due to global framework updates. Yay, right?

> .NET Core 2 brings back the concept of shared system runtimes to bring down the size of deployed applications without compromising application level runtime versioning.

.NET Core 1: Fragmentation and Deployment Size

Well, you win some and you lose some. With Version 1, the side-effect was that the .NET Core and ASP.NET frameworks were fragmented into a boatload of tiny, very focused NuGet packages that had to be referenced explicitly in every project. These focused packages are a boon to the framework developers as they allow for nice feature isolation and testing, and the ability to rev versions independently for each component.

.NET Core and .NET Standard

.NET Standard 2.0 is a **specification**, or a blueprint of features that have to be implemented by a specific platform. .NET Core 2.0 is a specific platform that **implements** the .NET Standard 2.0, which means that it has all the APIs required by Standard. Because .NET Core 2.0 implements .NET Standard 2.0, it can consume any libraries that are targeted to .NET Standard 2.0 or earlier. They're backward compatible.

.NET Standard includes a fixed number of APIs, which any platform that wants to implement the specification **has to** implement. The idea is that you have many concrete platforms that **support** .NET Standard and any library that implements that version of the Standard—or an earlier one—can then **run on any of these platforms**. This is great for library developers who can now build libraries that run on most of the .NET Platforms, including mobile platforms like Xamarin or other platforms like Xamarin.Mac.

But the result of all this micro-refactoring was that you had to add every tiny little micro-refactored NuGet Package/Assembly explicitly to each project. Finding the right packages to include was a big usability problem for application and library developers.

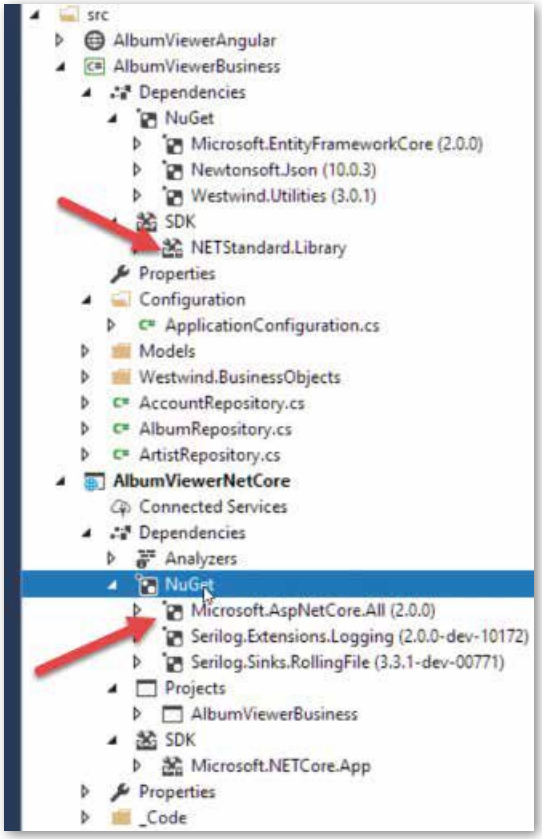


Figure 1: Package references are manageable again in V2.0.

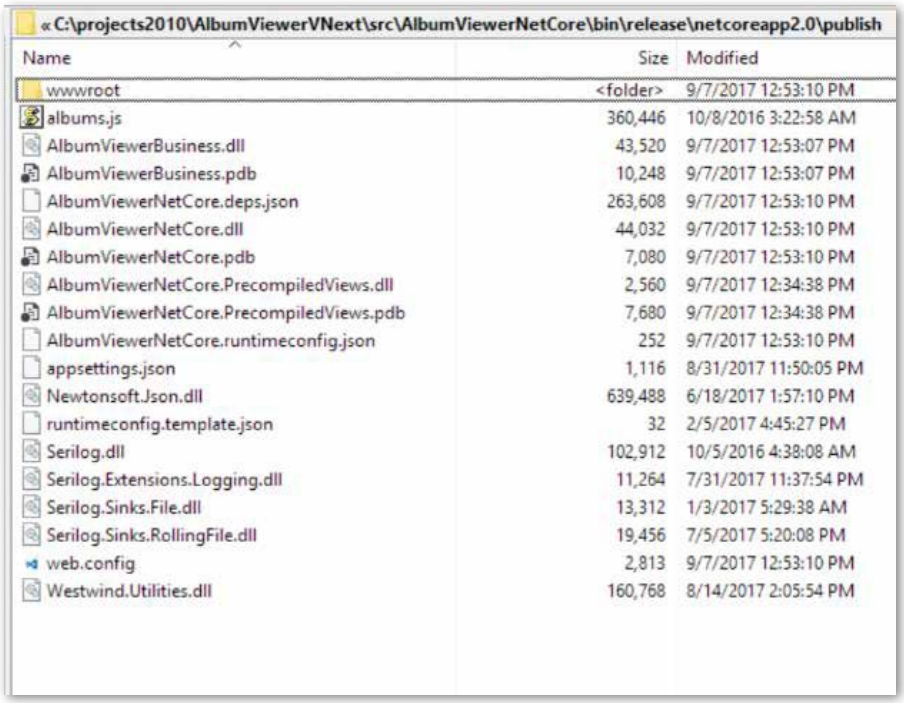


Figure 2: Published output contains only your code and your explicit dependencies

Additionally, when you published Web projects, all of those framework files—plus all runtime dependencies—had to be copied to the server, making for a huge payload to send up to a server for publishing even for a small HelloWorld Web application.

Meta Packages in 2.0

In .NET Core 2.0 and ASP.NET 2.0, this is addressed with system-wide Framework Meta Packages that can be referenced by an application. These packages are installed using either the SDK install or a runtime installer and can then be referenced from within a project as a single package. When you reference .NET Core App 2.0 in a project, it automatically includes a reference to the .NET Core App 2.0 meta package. Likewise, if you have a class library project that references .NET Standard 2.0, that meta package with all the required .NET Framework libraries is automatically referenced. You don't need to add any of the micro-dependencies to your project. The runtimes reference **everything** in the runtime, so in your code, you only need to apply namespaces but no extra packages or references.

There's also an ASP.NET Core meta package that provides **all of the ASP.NET Core and Entity Framework** references. Each of these meta packages have a very large predefined set of packages that are automatically referenced and available to your project's code.

This means that .NET Core projects no longer pre-load assemblies on startup, so if you don't use an assembly, it's never loaded and the runtime JIT compiler ensures that only the code you reference is compiled and loaded into memory. Even though you reference everything and the kitchen sink, your application only loads what it needs at runtime.

In your projects, this means you can reference .NET Standard in a class library project and get references to all the APIs that are part of .NET Standard with a **NetStandard.Library** reference, as shown in **Figure 1**. In applications, you can reference **Microsoft.NETCoreApp,** which is a reference to .NET Core 2.0. Here you specify an instance of runtime for the top-level application. For ASP.NET, the **Microsoft.AspNetCore.All** package brings in all ASP.NET and Entity Framework-related references in one simple reference.

Figure 1 shows an example of a two-project solution that has an ASP.NET Core Web app and a .NET Standard targeted business logic project:

Notice that the project references look very clean overall; I only explicitly add references to third-party NuGet packages. All of the system references come in automatically via the single meta package. This is even less cluttered than a full framework project, which still needed some high-level references. Here everything is automatically available for referencing.

This also is nice for tooling that needs to find references (Ctrl-. in VS or Alt-Enter for Resharper). Because **everything** is essentially referenced, Visual Studio or OmniSharp can easily find references and namespaces and inject them into your code as **using** statements. Nice!

Listing 1: The new, simpler SDK Style Project Format

```
<Project Sdk="Microsoft.NET.Sdk.Web">
  <PropertyGroup>
    <TargetFramework>netcoreapp2.0</TargetFramework>
    <UserSecretsId>d900d6cb-0e21-403b-94a3-17412045e7b4
    </UserSecretsId>
    <Version>2.0.0</Version>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.All"
                      Version="2.0.0" />
   <ProjectReference Include="..\AlbumViewerBusiness\AlbumViewerBusiness.
csproj" />
  </ItemGroup>

  <ItemGroup>
    <Content Update="wwwroot\**\*;Areas\**\Views;_
                    appsettings.json;album.js;web.config">
      <CopyToPublishDirectory>PreserveNewest</CopyToPublishDirectory>
    </Content>
  </ItemGroup>
  <ItemGroup>
    <Content Include="albumsdata.json">
      <CopyToOutputDirectory>PreserveNewest</CopyToOutputDirectory>
    </Content>
  </ItemGroup>
  <ItemGroup>
    <DotNetCliToolReference Include="Microsoft.DotNet.Watcher.Tools"
                            Version="2.0.0" />
  </ItemGroup>
</Project>
```

Runtimes++

In a way, these meta packages feel like classic .NET runtime installations. Microsoft now provides .NET Core and ASP.NET Core runtimes that are installed from the .NET Download site (https://www.microsoft.com/net/download) and can either be installed as only the runtime package or the full .NET SDK that includes all the compilers and command line tools so you can build and manage a project.

You can install multiple runtimes side-by-side and they're available to many applications to share. This means that the same packages don't have to be installed over and over for each and every application, which makes deployments a heck of a lot leaner than in 1.x applications.

You can still fall back to local packages installed with the application's output folder and override global installed packages, so now you get the best of all worlds. You can:

- Run an application with a pre-installed Runtime (default).
- Run an application with a pre-installed Runtime and override packages locally.
- Run an application entirely with locally installed Runtime packages.

In short, you now get to have your cake and eat it too, as you get to choose **exactly** where your runtime files are coming to.

Publishing Applications

Using ASP.NET Core 2.0, publishing a Web application to a Web server is much leaner than in prior versions. By default, using shared runtimes, the publish folder contains only **your compiled code** plus any explicit third-party dependencies from NuGet that you added to the project. You're no longer publishing runtime files to the server so the behavior is once again similar to what you had in a full framework Web application deployment.

Figure 2 shows the publish folder of the Solution shown in **Figure 1**.

This means that publishing your application is no longer a 100mb ordeal, but rather publishes a few megs of the stuff you actually built and use yourself. It's still possible to deploy full runtimes just as you could in v1 releases by explicitly using the **dotnet publish** command line and specifying a specific runtime version to install with, but by default, the publish process uses the pre-installed runtimes.

.NET SDK Projects

One of the nicest features of 2.0 (initially introduced in 1.1) is the new, SDK style **.csproj** project format. This project format is very lean and easily readable and maintainable, which is quite in contrast to the verbose and cryptic older **.csproj** format.

For example, it's not hard to glean what's going on in the **.csproj** file shown in **Listing 1**.

Notice that explicit file inclusion references are all but gone in the project file. Projects now assume that **all files are included** except those you explicitly exclude, which drastically reduces the content of a project file. The other big benefit is that you can simply drop files in the project folder to become part of the project; you no longer have to add files to a project explicitly.

Compared to the morass that was the old **.csproj** format, this makes it manageable to manually edit the project file as well as improving source control management, as there are much fewer changes than before.

> .NET SDK projects make it very easy to create a .NET project that targets multiple .NET runtimes and then creates a NuGet package.

Additionally, the new project format supports multi-targeting to multiple .NET Framework versions. I've talked about porting existing libraries to .NET Standard, and using the new project format, it's quite easy to set up a library to target both .NET 4.5 and .NET Standard, for example.

Listing 2 shows a truncated example of the Westwind.Utilities library that targets both .NET Standard and .NET 4.5 (full triple-target version: https://goo.gl/x4L3Kd).

Listing 2: A multi-targeted SDK Style Project

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFrameworks>netstandard2.0;net45</TargetFrameworks>
  </PropertyGroup>

  <PropertyGroup Condition="'$(Configuration)'=='Debug'">
    <DefineConstants>TRACE;DEBUG;</DefineConstants>
  </PropertyGroup>

  <PropertyGroup Condition=" '$(Configuration)' == 'Release' ">
    <DefineConstants>RELEASE</DefineConstants>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Newtonsoft.Json" Version="10.0.3" />
  </ItemGroup>

  <ItemGroup Condition=" '$(TargetFramework)' == 'netstandard2.0'">
    <PackageReference Include="System.Data.SqlClient"
                      Version="4.4.0" />
  </ItemGroup>
  <PropertyGroup
      Condition=" '$(TargetFramework)' == 'netstandard2.0'">
    <DefineConstants>NETCORE;NETSTANDARD;NETSTANDARD2_0
      </DefineConstants>
  </PropertyGroup>

  <ItemGroup Condition=" '$(TargetFramework)' == 'net45' ">
    <Reference Include="mscorlib" />
    <Reference Include="System" />
    <Reference Include="System.Core" />
    <Reference Include="Microsoft.CSharp" />
    <Reference Include="System.Data" />
    <Reference Include="System.Web" />
    <Reference Include="System.Drawing" />
    <Reference Include="System.Security" />
    <Reference Include="System.Xml" />
    <Reference Include="System.Configuration" />
  </ItemGroup>
  <PropertyGroup Condition=" '$(TargetFramework)' == 'net45'">
    <DefineConstants>NET45;NETFULL</DefineConstants>
  </PropertyGroup>
</Project>
```

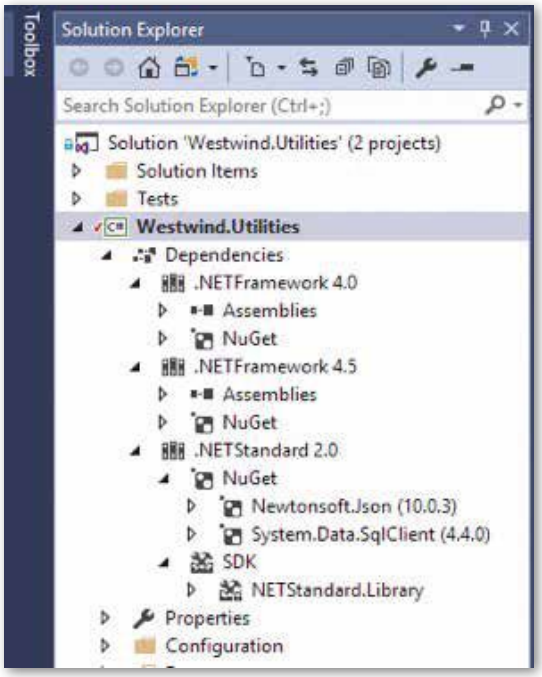


Figure 3: Multiple targets displayed in Visual Studio

The project defines two framework targets (this line is randomly broken to accommodate the columns in the magazine pages):

```
<TargetFrameworks>netstandard2.0;net45</TargetFrameworks>
```

Then it uses conditional target framework filtering to add dependencies. Visual Studio can visualize these dependencies for each target as well, as shown in **Figure 3** (with an additional **net40** target).

Visual Studio 2017.3+ also has a new Target drop down that lets you select which target is currently used to display code and errors in the environment. **Figure 4** shows how Visual Studio visualizes code given a specific runtime target.

There are other features in Visual Studio that make it target-aware:

- IntelliSense shows warnings for APIs that don't exist on any given platform.
- Compiler errors now show the target platform for which the error applies.
- Tests using the Test Runner respect the active target platform (VS2017.4+).

When you compile this project, the build system automatically builds output for all targets, which is very nice if you've ever tried to create multi-target projects with the old project system (hint: it sucked!).

The project system can also create a NuGet Package that wraps up all targets into one NuGet package. If you look back at the project file, you'll note that the NuGet Properties are now stored as part of the **.csproj** file.

Figure 5 shows what the build output from my three-target project looks like.

This is pretty awesome and frankly, something that should have been supported a long time ago in .NET!

Easier ASP.NET Startup Configuration

Another nice improvement and a sign of growing up is that the ASP.NET Core startup code in 2.0 is a lot more streamlined. There's simply quite a bit less of it.

The **absolute minimal ASP.NET Web application** you can build is just a few lines of code:

```
public static void Main(string[] args)
{
       WebHost.Start(async (context) =>
       {
```

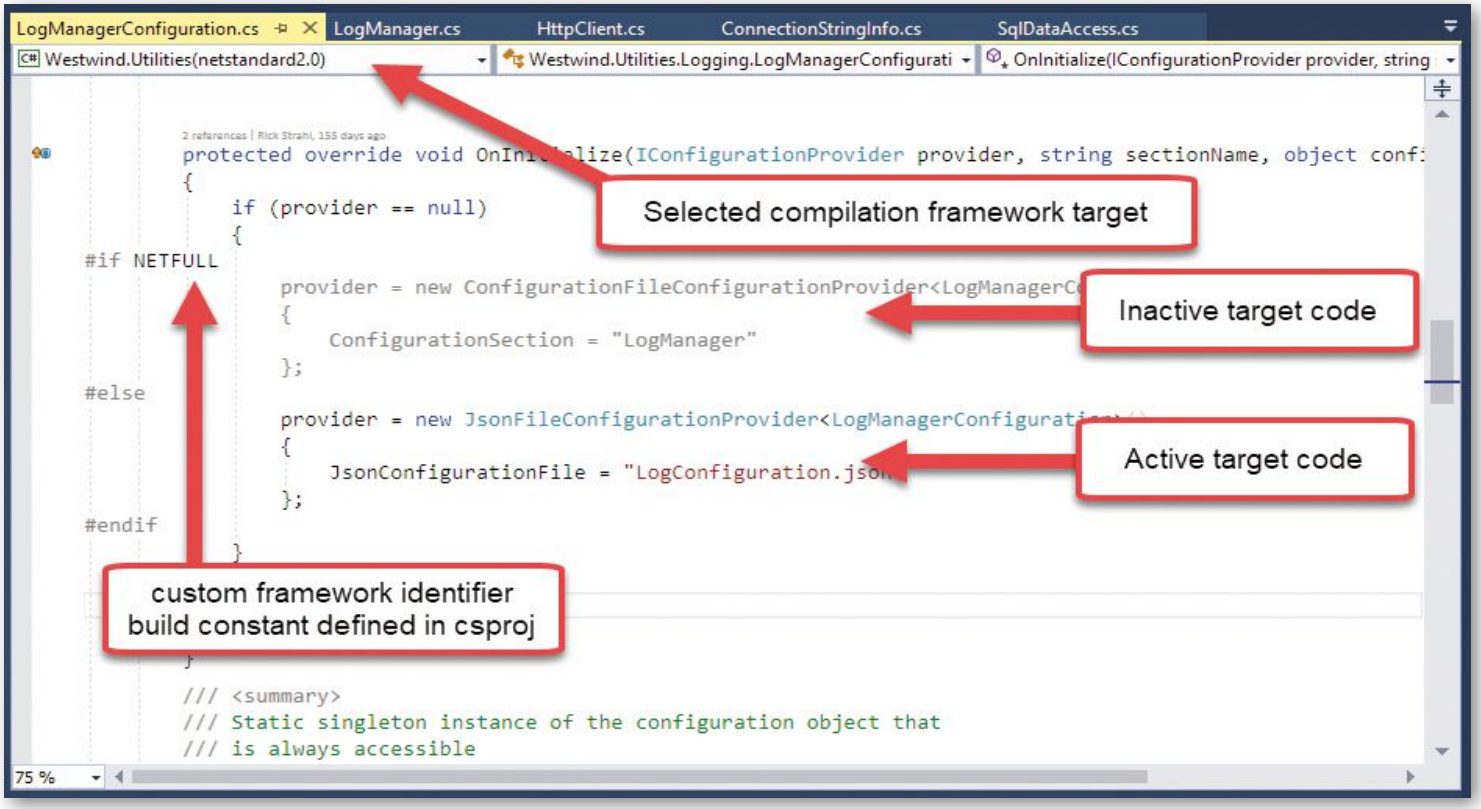


Figure 4: Visual Studio can detect the active target and highlight code accordingly.

```
        var r = $"Hello, time is: {DateTime.Now}";
        await context.Response.WriteAsync(r);
    })
    .WaitForShutdown();
}
```

Notice that this code works without any dependencies whatsoever, and yet has access to an **HttpContext** instance—there's no configuration or additional set up required, and the framework now uses a set of common defaults for bootstrapping an application. Hosting options, configuration, logging, and a few other items are automatically set with common defaults, so these features no longer have to be explicitly configured unless you want to change the default behavior.

The code also automatically hooks up hosting for Kestrel and IIS, sets the startup folder, allows for host URL specification and provides basic configuration features—all without any custom set up code. All of these things needed to be configured explicitly previously. Now, all of that is optional. Nice!

To be realistic though, if you build a real application that requires configuration, authentication, custom routing, CORS etc., those things must still be configured and obviously, that adds code. But the point is that ASP.NET Core now has a default configuration that out-of-the-box lets you get stuff done without doing any configuration.

The most common configuration set up looks like this:

```
public static void Main(string[] args)
{
   var host = WebHost.CreateDefaultBuilder(args)
        .UseUrls()
        .UseStartup<Startup>()
        .Build()
        .Run()
}
```

Using a Startup configuration class that handles minimal configuration for an MVC/API application looks like this:

```
public class Startup
{
      public void ConfigureServices(
                  IServiceCollection services)
      {
          services.AddMvc();
      }

      public void Configure(IApplicationBuilder app,
          IHostingEnvironment env,
          IConfiguration configuration)
      {
          app.UseStaticFiles();

          app.UseMvcWithDefaultRoute();
      }
}
```

You can then use either **RazorPages** (loose Razor files that can contain code) or standard MVC or API controllers to handle your application logic.

A controller, of course, is just another class you create that optionally inherits from Controller or simply has a **Controller** postfix:

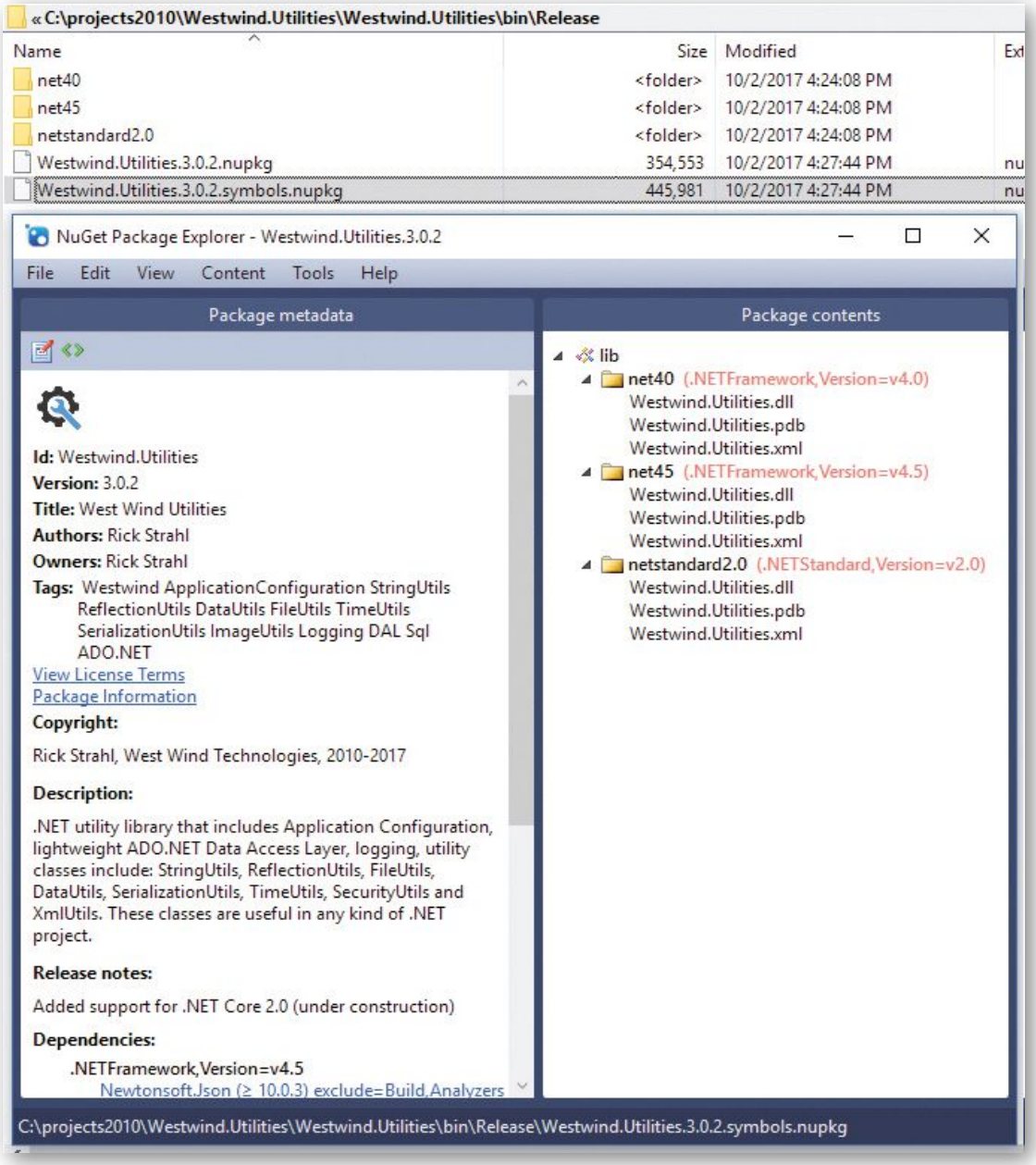


Figure 5: Multi-target projects automagically build for all target platforms and can create a NuGet package.

```
[Route("api")]
public class HelloController // : Controller
{
       [HttpGet("HelloWorld/{name}")]
       public object HelloWorld(string name)
       {
           return new
           {
               Name = name,
               Message = $"Hello World, {name}!",
               Time = DateTime.Now
           };
       }
}
```

In short, basic configuration for a Web application is now a lot cleaner than in 1.x versions.

One thing that has bugged me in ASP.NET Core is the dichotomy between the **ConfigureServices()** and **Configure()** methods. In 1.x, ASP.NET Core seemed to have a personality crisis about where to put configuration code for various components. Some components configured in **ConfigureServices()** using the **AddXXX()** methods, others did it in the **Configure()** method using the **UseXXX()** methods. In v2.0, Microsoft seems to have moved most configuration behavior into **ConfigureServices()** using options objects via Action delegates that get called later in the pipeline, so now things like CORS, Authentication, and Logging all use a similar configuration patterns.

For example, in the code in **Listing 3**, DbContext, Authentication, CORS, and Configuration are all configured in the **ConfigureServices()** method.

Listing 3: Configuration is a lot more organized in ASP.NET Core 2.0

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<AlbumViewerContext>(builder =>
    {
      var connStr = Configuration["Data:SqlServerConnectionString"];
      builder.UseSqlServer(connStr);
    });

    services
      .AddAuthentication(
                CookieAuthenticationDefaults.AuthenticationScheme)
          .AddCookie(o =>
          {
           o.LoginPath = "/api/login";
           o.LogoutPath = "/api/logout";
          });

    services.AddCors(options =>
    {
      options.AddPolicy("CorsPolicy",
        builder => builder
          .AllowAnyOrigin()
          .AllowAnyMethod()
          .AllowAnyHeader()
          .AllowCredentials());
    });

    // Add Support for strongly typed Configuration and map to class
    services.AddOptions();
    var sec = Configuration.GetSection("Application");
    services.Configure<ApplicationConfiguration>(sec);
}

public void Configure(IApplicationBuilder app)
{
       app.UseAuthentication();
       app.UseCors("CorsPolicy");
       app.UseDefaultFiles();
       app.UseStaticFiles();

       app.UseMvcWithDefaultRoute();
}
```

The **Configure()** method only enables the behaviors configured in ConfigureServices() by using various **.UseXXXX()** methods like **.UseCors("CorsPolicy")**, **.UseAuthentication()**, and **.UseMvc()** etc.

Although this still seems very disjointed, at least the patterns provided by Microsoft keep configuration logic mostly in a single place in **ConfigureServices()**.

I've been struggling with this same issue in porting another library, **Westwind.Globalization,** to .NET Core 2.0 and I needed to decide how to configure my component. I chose to follow the same pattern as Microsoft using **ConfigureServices()** with an Action delegate that handles option configuration:

```
services.AddWestwindGlobalization(opt =>
{

  opt.ResourceAccessMode =
        ResourceMode.DbResourceManager;
  opt.ConnectionString =config.ConnectionString;
  opt.ResourceTableName = "localizations ";
  opt.ResxBaseFolder = "~/Properties/";

  opt
 .ConfigureAuthorizeLocalizationAdministration(
          actionContext => return true);
});
```

This was implemented as an extension method with an Action delegate:

```
public static IServiceCollection
                  AddWestwindGlobalization(
   this IServiceCollection services,
   Action<DbResConfiguration> setOptionsAction)
{
       // add additional services to DI
       // configure based on options passed in
}
```

I'm not a fan of this (convoluted) pattern of indirect referencing and deferred operation, especially given that **ConfigureServices()** seems like an inappropriate place for component configuration when there's a **Configure()** method where I'd expect to be doing any configuring.

But once you understand how Microsoft uses the delegate-option-configuration-pattern, and if you can look past the consistent inconsistency, it's easy to implement and work with, so I'm not going to rock the boat and do something different in my components.

IRouterService: Minimal ASP.NET Applications

MVC or API applications are typically built using the MVC framework. As you've seen above, it's a lot easier with 2.0 to get an API application configured and up and running. But MVC has a bit of overhead internally.

> In 1.x, ASP.NET Core seemed to have a personality crisis about where to put configuration code for various components.

If you want something even simpler, perhaps for a quick one-off minimal microservice, or you're a developer who wants to build a custom framework on top of the core ASP.NET middleware pipeline, you can now do that pretty easily by taking advantage of **IRouterService,** which provides access only to the raw HTTP context bits, but no MVC processing framework. You can think of this approach as somewhat similar to an HttpHandler in classic ASP.NET.

Listing 4 shows another very simple single file, a self-contained ASP.NET application that returns a JSON response off a routed request.

Implementing .NET Standard 2.0

There are **many** .NET Standard 2.0 implementations: .NET Core 2.0, .NET 4.61, .NET 4.71, Xamarin for iOS 10.14 all **implement** .NET Standard 2.0. Note that these are very **specific** versions. Each of those platforms can consume a library that was built for .NET Standard 2.0 (or earlier) as well as platform-specific libraries for the specific platform. A .NET 4.61 application can consume both .NET Standard packages as well as .NET 4.5 assemblies, for example, and .NET Core 2.0 applications can use .NET Standard 2.0 and .NET Core assemblies, although I suspect that will be rather rare.

When you build class libraries going forward, you'll want to target .NET Standard to make the library available on as many platforms as possible. Top-level application projects such as an ASP.NET Core project, .NET Core, or full Framework Console apps will target a specific runtime. Because that runtime likely implements .NET Standard 2.0, it can then access any of the .NET Standard libraries.

The key is the **IRouterService** passed to the **app.UseRouter()**, which lets you directly map URLs to action delegates that have a request and a response you read from and write to. You can also access **HttpContext** from the **request** or **response** parameters so that all of the context's intrinsic objects are available. This is obviously a bit lower-level than using MVC/API controllers. There's no Controller logic available, so you have to handle input deserialization and output generation in your own code. The code above handles its own JSON serialization, for example. With a few simple helper extension methods, you can provide a lot of functionality using just this very simple mechanism. This is useful for publishing simple one-off "handlers." It can also be a good starting point if you ever want to build your own custom not-MVC Web framework or custom service.

IRouterService functionality is primarily for specialized use cases where you need one or more very simple notification request. It's very a similar use case to where you might employ server-less Web Functions (like Azure Functions or AWS Lambda) for handling simple service

Listing 4: A self-contained ASP.NET Core App using IRouterService

```
public static class Program
{
public static void Main(string[] args)
{
  WebHost.CreateDefaultBuilder(args)

    .ConfigureServices(s => s.AddRouting())
    .Configure(app => app.UseRouter(r =>
    {
      r.MapGet("helloWorldRouter/{name}",
        async (request, response, routeData) =>
        {
          var result = new
          {
            name = routeData.Values["name"] as string,
            time = DateTime.UtcNow
          };
          await response.Json(result);
        });
      r.MapPost("helloWorldPost"
                async (request, response, routeData) => {
        ...
      };
    }))
    .Build()
    .Run();
}

public static Task Json(this HttpResponse response, object obj,
                        Formatting formatJson = Formatting.None)
{
  response.ContentType = "application/json";

  JsonSerializer serializer = new JsonSerializer
    {
     ContractResolver = new CamelCasePropertyNamesContractResolver()
    };
  serializer.Formatting = formatJson;

  using (var sw = new StreamWriter(response.Body))
  using (JsonWriter writer = new JsonTextWriter(sw))
  {
    serializer.Serialize(writer, obj);
  }

  return Task.CompletedTask;
}
}
```

Listing 5: RazorPages lets you embed

```
@model IndexModel
@using System.ComponentModel.DataAnnotations
@using Microsoft.AspNetCore.Mvc.RazorPages
@functions {

  public class IndexModel : PageModel
  {

    [MinLength(2)]
    public string Name { get; set; }

    public string Message { get; set; }

    public void OnGet()
    {
      Message = "Getting somewhere";
    }


    public void OnPost()
    {
      TryValidateModel(this);

      if (this.ModelState.IsValid)
        Message = "Posted all clear!";
      else
        Message = "Posted no trespassing!";
    }
  }
}
@{
  Layout = null;
}
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<form method="post" asp-antiforgery="true">
  <input asp-for="Name" type="text" />
  <button type="submit">Show Hello</button>
  @Model.Name
</form>

<div class="alert alert-warning">
  @Model.Message
</div>
</body>
</html>
```

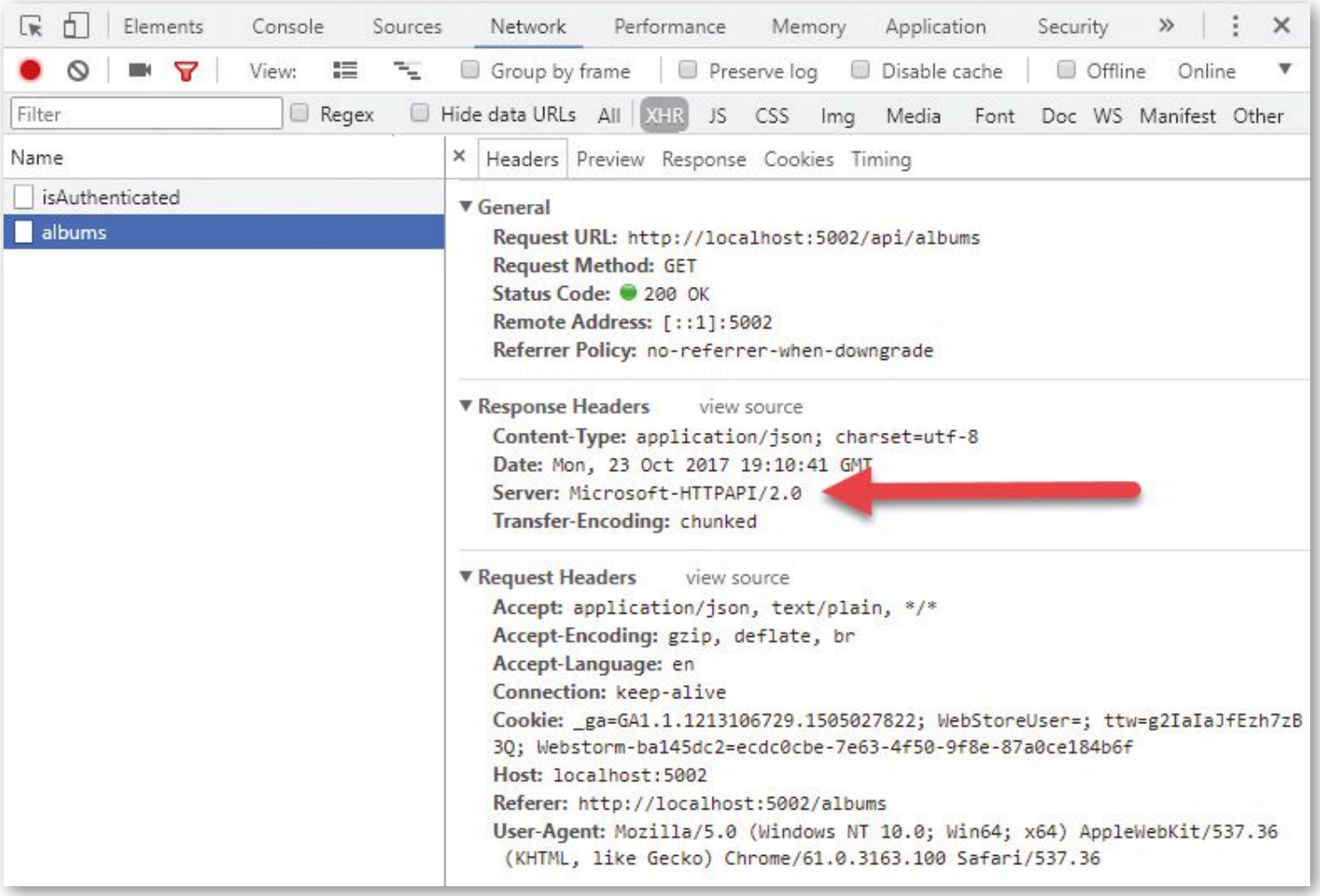


Figure 6: Http.sys hosting in ASP.NET Core provides efficient Windows hosting without a proxy

callbacks or other one-off operations that have few dependencies.

I've also found **IRouterService** useful for custom route handling that doesn't fall into the application space, but is more of an admin feature. For example, recently I needed to configure an ASP.NET Core app to allow access for Let's Encrypt's domain validation callbacks and I could just use a route handler to handle a special route in the server's **Configure()** code:

IRouterService makes it easy to create simple, routed Delegates that respond to Requests without using MVC.

```
app.UseRouter(r =>
{
  r.MapGet(".well-known/acme-challenge/{id}",
  async(request, response, routeData) =>
  {
    var id = routeData.Values["id"] as string;
    var file = Path.Combine(env.WebRootPath,
      ".well-known", "acme-challenge", id);
    await response.SendFileAsync(file);
  });
});

app.UseMvcWithDefaultRoute();
```

This code handles a very specific URL pattern inline as part of the configuration code. I could have also handled this with a custom MVC route, but it's arguably easier and cleaner to set up a custom IRouterService route to handle a very specific, non-application concern like this as part of the application configuration process.

Http.Sys Support

For Windows, ASP.NET Core 2.0 now also supports Http.sys as another Web server in addition to the Kestrel and IIS/IIS Express servers that are supported by default. **Http.sys** is the kernel driver used to handle HTTP services on Windows. It's the same driver that IIS uses for all of its HTTP interaction, and now you can host your ASP.NET Core applications directly on Http.sys using the **Microsoft.AspNetCore.Server.HttpSys** package.

The advantage of using Http.sys directly is that it uses the Windows kernel HTTP infrastructure, which is a hardened Web Server front-end that supports high-level sup-

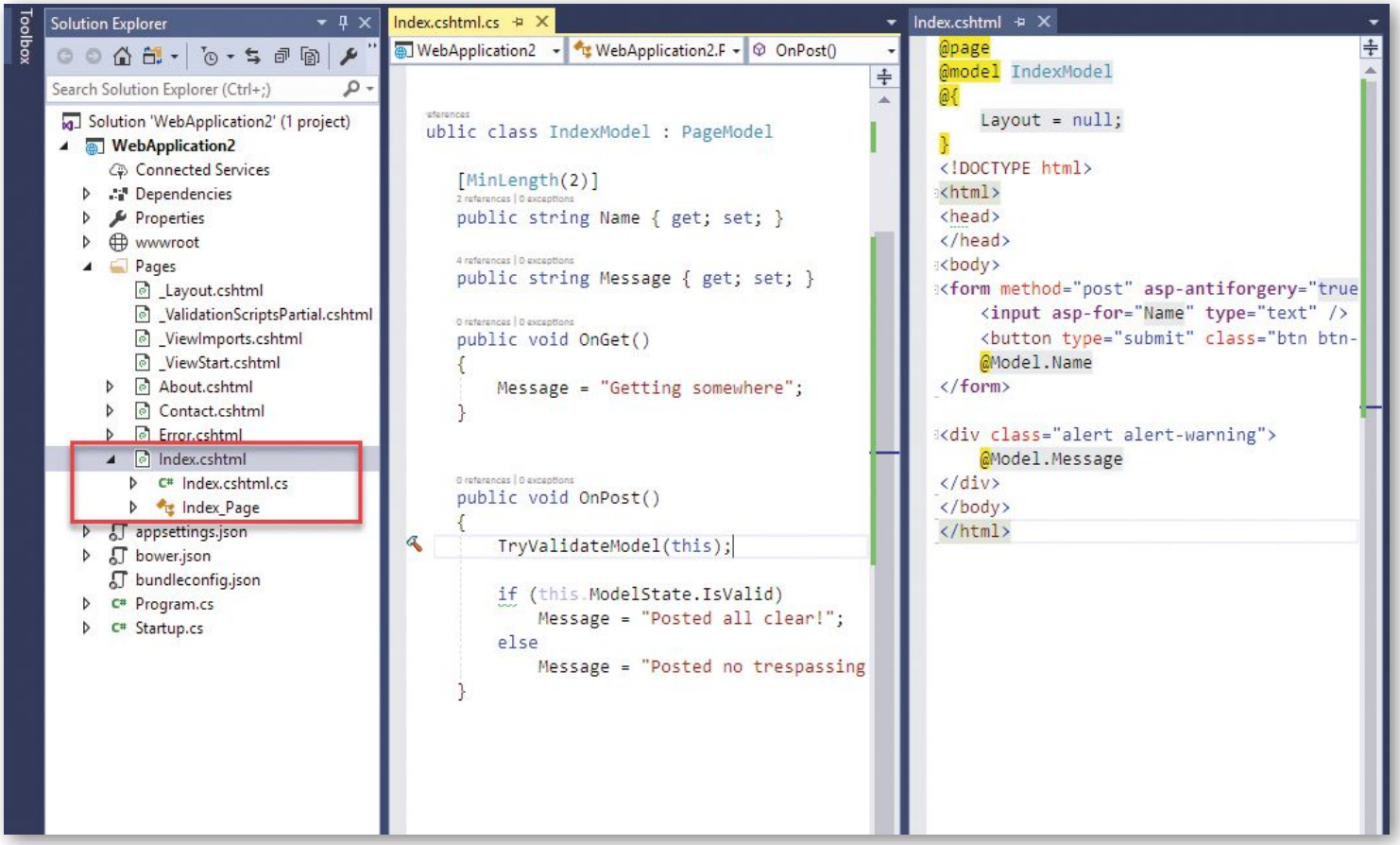


Figure 7: RazorPage Code-Behind uses a hybrid model/controller PageModelclass

.NET Core Runtimes

With .NET Core 2.0, Microsoft decided to go back to distributing system-level runtimes that are pre-installed much in the way that the full .NET Framework was installed in the past.

The big difference with the .NET Core runtimes is that each runtime is versioned and installed side-by-side with other versions. Each version can exist independently of any other version, so there are no version conflicts.

This differs from full framework runtimes, which were rev'd very infrequently and are updated in place, resulting in potential version issues.

Applications can specify exactly which version of the .NET Core runtime they want to target when the application is built and assuming that the runtime is installed on the local or deployed computer, the application can then run using that runtime.

port for SSL, content caching and many security-related features not currently available with Kestrel.

For Windows, the recommendation has been to **use IIS as a front-end reverse proxy** in order to provide features like static file compression and caching, SSL management, and rudimentary connection protections from various HTTP attacks against the server.

> RazorPages provide a new approach for creating scripted server-side HTML content, using an approach similar to client-side frameworks like Angular or Vue.

By using the Http.sys server, you can get most of these features without having to use a reverse proxy in front of Kestrel, which has a bit of overhead.

To use Http.sys, you need to explicitly declare it using the **.UseHttpSys()** configuration added to the standard startup sequence (in **program.cs**):

```
WebHost.CreateDefaultBuilder(args)
       .UseStartup<Startup>()
       .UseHttpSys(options =>
       {
         options.Authentication.Schemes =
           AuthenticationSchemes.None;
         options.Authentication
           .AllowAnonymous = true;
         options.MaxConnections = 100;
         options.MaxRequestBodySize = 30000000;
         options.UrlPrefixes
           .Add("http://localhost:5002");
       })
       .Build();
```

You can then configure the local port to make it accessible both locally and remotely (by opening up a port on the firewall). When you do, you'll see the HTTP Sys server as part of the response headers, as shown in **Figure 6**.

Microsoft has a detailed document that describes how to set up http.sys hosting at https://goo.gl/XnFz88.

If you're running on Windows, this might be a cleaner and more easily configurable way to run ASP.NET Core applications than doing the Kestrel > IIS dance. Doing some quick over-the-finger performance tests with **West Wind WebSurge** (websurge.west-wind.com) shows that running with raw Http.sys is a bit faster than running Kestrel with IIS as a front-end proxy because it eliminates the Proxy hop. However, by cutting out IIS, your application is now responsible for serving all content rather than letting IIS handle standard static content.

For a public-facing website, you're probably better off with full IIS, but for raw APIs or internal applications, Http.sys is a great alternative for Windows-hosted server applications.

RazorPages

In ASP.NET Core 2.0, Microsoft rolls out **RazorPages**. RazorPages is completely new, although it's based on concepts that should be familiar to anybody who's used either ASP.NET WebPages or—gasp—even WebForms that came before it.

When I first heard about RazorPages a while back, I had mixed feelings about the concept. Although I think that a script-based framework is an absolute requirement for many websites that deal primarily with content, I also feel that requiring a full ASP.NET Core application setup with controllers and views separated out, plus a full site deployment process just to run script pages is a bit of overkill. One of the advantages of tools like WebPages and WebForms is that you don't have to "install" an application and you just drop a new page into a server and run. With RazorPages, you're still installing and deploying a Web "application."

But think about how much clutter is involved in MVC to get a single view fired up in the default configuration that ASP.NET MVC projects use. There's:

- Controller Class
- Controller Method that invokes your View (Controllers folder)
- View Model (Models Folder)
- View (View/Controller folder)

In other words, code in MVC is scattered all over the place and there's quite a bit of ceremony to get a page rendered. MVC works great for "real" applications that require extensive logic to display pages. But MVC is terrible for mostly static content pages that only have minimal server logic embedded in pages like landing pages, company or product and news sites, etc. that require a little bit of dynamic logic. This type of content is much better served by self-contained script pages.

RazorPages mostly addresses this scenario, but at the same time, it's also powerful enough to create a much more complex application with an optional fresh new PageModel class that allows attaching multiple actions based on HTTP verbs. This optional Code-Behind-like approach has common startup code and specifics for GET and POST operations, for example. This is a different way of doing MVC that has much more in common with client-side frameworks like Angular or Vue than ASP.NET MVC.

One thing to understand is that RazorPages still requires an ASP.NET Core application to be deployed on the server. Unlike WebForms or WebPages, which **just work** by dropping a page into a virtual folder or website, RazorPages are hosted inside of their own ASP.NET Core application, which means you still have to build and deploy that application. Once the application is up, you can simply add or update Razor Pages while the application is running and as Razor Pages are dynamically compiled at runtime. Because Razor Pages is part of an application, it also provides the benefits of being able to create custom components and take part of the ASP.NET Core Middleware pipeline and everything else that ASP.NET Core's base functionality provides. It's up to you to decide what you want to support.

RazorPages use the same exact Razor syntax that MVC uses, so unlike the older WebPages technology, RazorPages and MVC Views share the same Razor engine and parsers and you can move Views between the two models easily. If you really want to get fancy, you can even mix and match and use RazorPages and MVC in the same application. A common scenario might be to use RazorPages for HTML content and MVC for API requests.

Listing 5 shows an example of a self-contained scripted Razor page that includes a model and controller methods inline.

Before you completely dismiss inline code in the **.cshtml** template, consider that code inside the RazorPage is dynamically compiled at runtime, which means that you can make changes to the page without having to recompile and restart your entire application as you have to do with controller or even Code-Behind code in ASP.NET Core applications.

Although I really like the fact that you can embed a model right into the Razor page for simple pages, this gets messy quickly. More commonly, you pull out the **PageModel** into a separate Code-Behind PageModel class. The default template that creates a RazorPage in Visual Studio does just that. When you create a new RazorPage in Visual Studio, you get a **.cshtml** and a nested **.cshtml.cs** file, as shown in **Figure 7**.

The **PageModel** subclass in this scenario becomes a hybrid of controller and model code very similar to the way many client-side frameworks like Angular handle the MV* operation, which is more compact and easier to manage than having an explicit controller located in yet another external class.

PageModel supports implementation of a few well-known methods, like **OnGet()**, **OnPost()**, etc., for each of the supported verbs that can handle HTTP operations just like you would in a controller. An odd feature called **PageHandlers** using the **asp-page-handler="First"** attribute lets you even further customize the methods that are fired with a method postfix like **OnPostFirst()**, so that you can handle multiple forms on a single page for example.

Although traditional MVC feels comfortable and familiar, I think RazorPages offers a viable alternative with page-based separation of concerns in many scenarios. Keeping View and View-specific Controller code closely associated usually makes for an easier development workflow and I'd already moved in this direction with feature folder setup in full MVC anyway. If you squint a little, the main change is that there are no more explicit multi-concern controllers, just smaller context specific classes.

RazorPages is not going to be for everyone, but if you're like me and initially skeptical, I encourage you to check them out. It's a worthwhile development flow to explore and for the few things I still use server-generated

HTML for, I think RazorPages will be my tool of choice on ASP.NET Core HTML content.

Tooling Issues Have Been Mostly Resolved

I've talked about a lot of things that are improved and that make 2.0 a good place to jump into the fray for .NET Core and ASP.NET Core. But it's not without its perils: there are still a few loose ends, especially when it comes to tooling.

When I wrote about problem issues three months ago in a blog post (https://goo.gl/qX8R6Z), there were major tooling issues with SDK projects that made working with .NET Core and .NET Standard projects pretty painful. Slow compiles, terribly slow run/debug/restart cycles, incorrect and non-clear error information in Visual Studio and other tools, slow .NET run watch recycles made for a pretty dreadful developer experience. I'm happy to say that most of the performance and stability issues I griped about in that original blog post have been addressed in subsequent SDK and Visual Studio updates.

For this reason, I highly recommend that you make sure you're running the latest version of the .NET SDK (2.12 at the time of writing) and Visual Studio 2017 (vs2017.5.5 at time of writing) or OmniSharp. If you've experienced these slow and unstable tooling issues and stepped away from .NET Core/Standard because you thought it was just too unstable, it might be time to give it another try. Faster and more reliable tooling makes a world of difference in how you experience the platform.

In the three months since my last blog post, performance of the tooling has improved drastically and overall stability of the compiler tools in Visual Studio, as well as other tools like Visual Studio Code using OmniSharp as well as JetBrains' new Rider IDE, is vastly better. Also gone are the constant invalid compiler errors and false-positive error messages. Tests now run reasonably fast and both Test Explorer and Resharper's Test Runner can now clearly display tests for multiple build targets.

The tooling is clearly on the right path to being on par with full framework tools in terms of stability and performance, but we're not quite there yet. There are still rough edges, but for now, I'd at least declare it as **good enough to go**.

Go Ahead and Jump

The entire 2.0 train of upgrades is a **huge** improvement over what came in V1.x, and the progress of change has been pretty rapid and amazing. I—and many others—were sitting on the fence with .NET Core/Standard for a long time. It was easy to see the promise of the .NET Core/Standard long ago, but execution and implementation in V1 left too many holes that made jumping in seem too daunting or simply not enough of an incentive. For v1.x, I was sitting on the fence, dabbling, watching, learning, and I have to say that I'm glad I waited out the initial releases before doing any real work.

But with v2.0, I've jumped in with both feet, converting old full-framework libraries to .NET Standard and ASP.NET Core versions, as well as starting a number of internal ASP.NET Core 2.0 projects. My first work has been around libraries, which has been a great learning experience, and the experience has been a good one. In fact, the benefits of the new project system and multi-targeting have been a big improvement over previous versions. I've also finally started a few ASP.NET Core Web projects for some internal projects that are in dire need of updates, and using many of the new features in ASP.NET Core has been a pure joy. There are still challenges because there are many changes in how things work and rediscovering where some features live (the "who moved my cheese" syndrome), but overall, the reimagining of the ASP.NET platform is a vast improvement.

Another thing that I find extremely promising is that Scott Hunter recently mentioned that .NET Core 2.0 and .NET Standard 2.0 will stay put for a while, without major breaking changes moving forward. I think this is a great call: I think we all need a breather instead of chasing the next pie in the sky. We can all use some time to try to catch up to the platform, figure out best practices, and also see what things still need improving.

It's been a rough ride getting to 2.0 and I, for one, can appreciate a little stretch of smooth road ahead.

Rick Strahl
CODE

Getting Started with Node Streams

Asynchrony is at the heart of NodeJS. Developers are told, "Don't block the event loop." NodeJS uses a form of cooperative multitasking that relies on code yielding frequently so that other code can run. Asynchrony presents an interesting challenge to overcome when writing code that would normally be synchronous: uncompressing a file, reading a CSV file, writing out a PDF file,

Chris Kinsman
@chriskinsman
chris@kinsman.net

Chris Kinsman is the Chief Architect of PushSpring, which offers accurate and relevant audience data for mobile ad targeting. He's a Microsoft Regional Director with a focus on startups.

or receiving a large response from an HTTP request. In an Express-based Web server, it would be a terrible idea to synchronously take an upload request, compress it, and write it to disk. Express won't be able to handle any other incoming HTTP requests from other clients while the upload is being processed.

Many of these challenges are answered by an abstract interface in NodeJS called a **stream**. You've probably worked with streams in Node and not known it. For example, process.stdout is a stream. A request to an HTTP server is a stream. All streams provide two ways to interact with them: Events or Pipelines.

Events

All streams are instances of EventEmitter, which is exposed by the Events module. An EventEmitter allows consuming code to add listeners for events defined by the implementer. Most Node developers encounter this pattern when looking for a way to read a file. For instance, createReadStream in the fs module returns a Readable stream that's an EventEmitter instance. You obtain the data by wiring up a listener on the **data** event like this:

```
const fs = require('fs');

const readStream = fs.createReadStream(
          'test.txt', {encoding: 'utf8'});

readStream.on('data', function (chunk) {
  console.log(chunk);
});
```

As the Readable stream pulls the data in from the file, it calls the function attached to the **data** event each time it gets a chunk of data. An **end** event is raised at the end of the file. (See a full example in the samples download folder called "Events".)

Problems quickly arise when using events. What if you want to take this data and do something else with it that's also asynchronous and slower than the data can be read? Perhaps you want to send it to an FTP server over a slow connection. If you call the second asynchronous service inside the event listener above and it yields to the event loop, you'll likely receive another call to your event listener before the data has finished being processed. Overlapped events can lead to out-of-order processing of the data, unbounded concurrency and considerable memory usage as the data is buffered. What you really need is a way to indicate to the EventEmitter that until the event listener is done processing the event you don't want another event to fire. This concept is frequently termed **backpressure**. In other parts of node, this is handled by requiring the event listener to make a callback to indicate that it's done. EventEmitter doesn't implement this pattern. EventEmitter does provide pause() and resume() methods to pause the emission of events.

Pipe offers a better alternative for reading data from a stream and performing an asynchronous task.

Pipe

All streams implement a pipeline pattern (as you can read in this interesting article: http://www.informit.com/articles/article.aspx?p=366887&seqNum=8). The pipeline pattern describes data flowing through a sequence of stages, as shown in **Figure 1**. There are three main players that you encounter in the pipeline pattern: a Readable source, a Writable destination, and optionally zero or more Transforms that modify the data as it moves down the pipeline.

As the chunks of data are read by the Readable stream, they're "piped" into another stream. Piping a Readable stream to a writable stream looks like this:

```
const fs = require('fs');

const readStream =
          fs.createReadStream('test.txt');
const writeStream =
          fs.createWriteStream('output.txt');
readStream.pipe(writeStream);
```

Data is read by the Readable stream and then pushed in chunks to the Writable stream. In the example above, how do you know when the contents of test.txt have all been written to output.txt? Just because you're using the pipe method doesn't turn off the events raised by the streams.

Figure 1: Pipeline Pattern

Events are still useful for knowing what's going on with a stream. To get a notification when all the data has passed through the stream, add an event listener like this:

```
const fs = require('fs');

const readStream =
          fs.createReadStream('test.txt');
const writeStream =
          fs.createWriteStream('output.txt');

writeStream.on('end', () => {
  console.log('Done');
});

readStream.pipe(writeStream);
```

Note that the event listener is wired up before calling the Pipe method. Calling Pipe starts the Readable stream. If you wire up events after calling Pipe, you may miss events that are fired before your listener is in place.

Use a Transform if you want to modify the data as it passes from Readable to Writable. Perhaps you have a compressed file that you need to decompress. Use the Gunzip transform provided in the zlib module like this to uncompress the data:

```
const fs = require('fs');
const zlib = require('zlib');

const readStream =
          fs.createReadStream('test.txt.gz');
const writeStream =
          fs.createWriteStream('output.txt');

writeStream.on('end', () => {
  console.log('Done');
});

readStream
          .pipe(zlib.createGunzip())
          .pipe(writeStream);
```

Calling zlib.createGunzip creates a transform stream to uncompress the data flowing through it. Note that the event listener is wired up on the Writable stream at the end of the pipeline. You could listen for the **end** event on the Readable stream. However, in cases where the Writable stream or Transform stream is slower than the Readable stream, it may be a considerable amount of time before all of the data is processed. In particular, because of the way it buffers data for efficient decompression, the Gunzip transform causes the **end** event to fire on the Writable stream much later than the **close** event fires on the Readable stream. In general, listening for **end** on the Writable stream is the right choice.

Creating Your Own Streams

Using the streams provided by node is a great start, but the real power of streams comes into play when you start to build your own streams. You can create streams to read a 4GB compressed file from a cloud provider, convert it into another format, and write it back out to a new cloud provider in a compressed format without it ever touching the disk. Streams allow you to decompose each chunk of that process in a fashion that provides re-usable pieces that you can plug together in different ways to solve similar but different problems.

There are two main ways to construct your own streams based on Node's built in stream module: inheritance and simplified construction. Simplified construction, as its name implies, is the easiest. It's especially nice for quick one-off transforms. Inheritance syntax is a bit more verbose but allows the definition of a constructor to set object-level variables during initialization. In my examples, I use the inheritance-based syntax.

Create a Readable Stream

Readable streams source data that are piped into downstream Transform or Writable streams. I don't create Readable streams nearly as often as I create Transform or Writable streams. A Readable stream can source its data from anywhere: a socket, a queue, some internal process, etc. For this article, I've created a simple Readable that streams bacon ipsum from an internal JSON data structure.

Creating a Readable stream is fairly simple. Extend the built-in Readable stream and provide an implementation for one method: **_read**.

```
class BaconReadable extends stream.Readable {
  constructor(options) {
    super(options);
    this.readIndex = 0;
  }

  _read(size) {
    let okToSend = true;
    while (okToSend) {
      okToSend = this.push(
                              baconIpsum.text.substr(
                              this.readIndex, size));
      this.readIndex += size;

      if (this.readIndex >
                              baconIpsum.text.length) {
        this.push(null);
        okToSend = false;
      }
    }
  }
}
```

The **baconIpsum.text** contains the text that the readable emits and **_read()** does the bulk of the work. It's passed an advisory size in bytes that indicates how much data should be read. More or less data than indicated by the size argument may be returned, in particular, if the stream has less data available than the size argument indicates, there's no need to wait to buffer more data; it should send what it has.

When **_read()** is called, the stream should begin pushing data. Data is pushed downstream by calling **this.push()**. When **push** returns false, which is a form of **backpressure,** the stream should stop sending data. Calling **push** doesn't immediately pass the data to the next stage in the pipeline. Pushed data is buffered by the underlying Readable imple-

mentation until something downstream calls **read**. To control memory utilization, the buffer isn't allowed to expand indefinitely. Controlling the buffer size is handled by the highWaterMark option that can be passed into the constructor of a stream. By default, the highWaterMark is 16KB, or for streams in objectMode, it's 16 objects. Once the buffer exceeds the highWaterMark, **push** returns false and the stream implementation shouldn't call **push** until **_read** is called again. Flow control like this is what allows streams to handle large amounts of data while using a bounded amount of memory.

Passing a null value to **push** is a special signal indicating that the Readable has no more data to read. When there's no more data to read, the **end** event on the Readable stream fires.

BaconReadable is used like the previous example's Readable methods:

```
const BaconReadable =
          require('../BaconReadable');
const fs = require('fs');

const baconReader = new BaconReadable();

const fileWriter =
          fs.createWriteStream('output.txt');

fileWriter.on('finish', () => {
    process.exit(0);
});

baconReader.pipe(fileWriter);
```

When run, this code writes 50 paragraphs of bacon ipsum to output.txt.

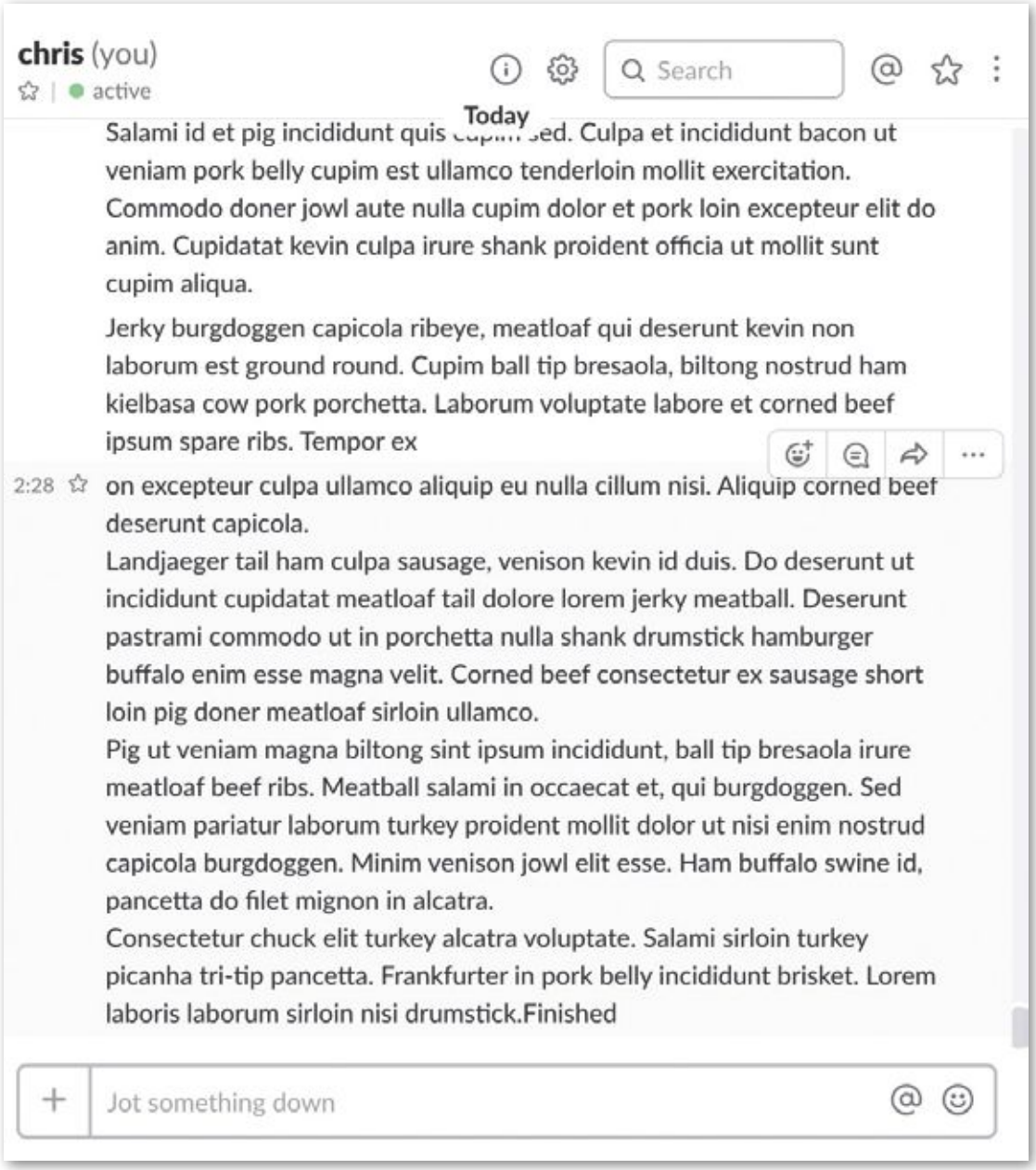


Figure 2: Bacon ipsum Slack output

Create a Writable Stream

Writable streams sink data at the end of a stream pipeline. I use them quite frequently to post the contents of a file to an HTTP endpoint or to upload data to a cloud storage service. The SlackWritable I show here posts data from the stream into a Slack channel. It's a perfect way to share some bacony goodness with your coworkers!

Creating a Writable stream follows a similar pattern to Readable. Extend the built-in Writable stream and implement a single method, **_write**.

```
const request = require('request');
const stream = require('stream');

class SlackWritable extends stream.Writable {
    constructor(options) {
        options = options || {};
        super(options);

        this.webHookUrl = options.webHookUrl;
    }

    _write(chunk, encoding, callback) {
        if (Buffer.isBuffer(chunk)) {
            chunk = chunk.toString('utf8');
        }

        request.post({ url: this.webHookUrl,
             json: true,
             body: {text: chunk}
        }, callback);
    }
}
```

The first thing to note in this example is the constructor. I've extended the default option implementation of the underlying stream and used it to pass in the webHookUrl for the slack integration. Just a reminder for those new to ES6: You can't set properties on **this** until after you have called the **super** method. I've left out the code ensuring that a webHookUrl is always passed in. See the SlackWritable example in the downloads for how to handle this.

The core of the implementation is **_write**. It receives three arguments: a chunk, the encoding, and a callback. Chunk is the data received from upstream. It can be in multiple formats including a buffer, a string, or an object. It'll almost always be a Buffer unless objectMode is set to **true**, in which case, it'll be an object. Because SlackWritable needs a string, it first checks to see if the chunk is a buffer. If it's a buffer, you convert the buffer to a string.

You may wonder why SlackWritable explicitly uses **utf8** instead of using the passed-in encoding variable. Encoding is only valid if the chunk is a string. If the chunk is a buffer, it should be ignored. Once the buffer has been converted to a string, it's then posted to slack using the request module. When the post has completed, the callback is called. Calling the callback indicates to the upstream that the data sent in via **_write** has been handled and that new data can be sent. Node won't call **_write** again until the previous write command has completed. If you combine the BaconReadable with your new SlackWritable, you get code that looks like this:

```
const BaconReadable =
          require('../BaconReadable');
```

Listing 1: Line Transform

```
const stream = require('stream');

class LineTransform extends stream.Transform {
    constructor(options) {
        options = options || {};
        super(options);

        this.separator = options.separator || '[\r\n|\n|\r]+';
        this.chunkRegEx = new RegExp(this.separator);
        this.remnantRegEx = new RegExp(this.separator + '$');

        this.remnant = '';
    }

    _transform(chunk, encoding, callback) {
        // Convert buffer to a string for splitting
        if (Buffer.isBuffer(chunk)) {
            chunk = chunk.toString('utf8');
        }

        // Prepend any remnant
        if (this.remnant.length > 0) {
            chunk = this.remnant + chunk;
            this.remnant = '';
        }

        // Split lines
        var lines = chunk.split(this.chunkRegEx);

        // Check to see if the chunk ends exactly
        //with the separator
        if (chunk.search(this.remnantRegEx) === -1) {
            // It doesn't so save off the remnant
            this.remnant = lines.pop();
        }

        // Push each line
        lines.forEach(function (line) {
            if (line !== '') { this.push(line); }
        }, this);

        return setImmediate(callback);
    }

    _flush(callback) {
        // Do we have a remnant?
        if (this.remnant.length > 0) {
            this.push(this.remnant);
            this.remnant = '';
        }

        return setImmediate(callback);
    }
}
```

Figure 3: Bacon ipsum Slack output broken up by paragraphs

```
const SlackWritable = require('../SlackWritable');

const baconReader = new BaconReadable();
const slackWriter = new SlackWritable(
          { webHookUrl: process.env.WEBHOOKURL});

slackWriter.on('finish', () => {
    process.exit(0);
});

baconReader.pipe(slackWriter);
```

The **webHookUrl** is passed in via an environment variable to avoid it being committed in code and inadvertently disclosed. If you run the code and pass it a valid slack **webHookUrl,** you'll see something like **Figure 2** in your Slack client.

If you look closely at the Slack output you'll notice that the bacon ipsum is broken up into **chunks** of text. These chunks do not align with the \n line feeds in the original text. There are two reasons for this:

- Slack breaks up all text sent via a webHook into chunks of < 4096 characters before sending.
- Twice, **_write** was called: once with a 16KB block of text and once with a 953-byte block of text.

The number 16KB should ring a bell. It matches the default highWaterMark size for buffering between streams. By changing this value, you could alter the size of each chunk that **_write** receives. What if you want **_write** to get called each time a line ending in **\n** appears in the stream? Transform streams to the rescue!

Creating a Transform Stream

Transform streams sit between Readable and Writable streams in the pipeline. You're not limited to one Transform stream. It's possible to string together multiple transforms. Taking advantage of this allows you to create Transforms with a single responsibility and re-use them

in multiple pipelines in various ways. You might imagine a Readable AWS S3 stream piped into a Gunzip Transform and then piped into a CSV parsing Transform piped into a Filter Transform that removes certain rows piped into a Gunzip Transform piped into a Writable AWS S3 stream. A pipeline built like this can handle a file of any size with no change in memory or disk requirements. A larger file would, of course, require more time/CPU to process.

You can easily create a Transform that breaks up the bacon ipsum into lines before sending it along to SlackWritable. Creating a Transform stream follows the well-worn pattern you've now established with Readable and Writable: extend the built-in Transform stream and implement the **_transform** method. See **Listing 1** for the complete Line Transform implementation

Let's examine the constructor first. An optional line separator is passed into the constructor as part of the options hash and if it doesn't exist uses CRLF, LF, or CR. It creates two regular expressions that are used to parse the text as it's transformed. Finally, it creates a string variable to hold partial lines, called **remnant**.

The bulk of the work is done by **_transform**. Similar to the Writable stream, it receives a chunk, an encoding, and a callback. If a buffer is received, it's first converted to a string. It checks for any remnants from the last transform call, prepends that data onto the chunk of data that was just received, and clears the remnant. It then splits the lines up using the separator regular expression. After splitting the lines, it checks to see if the last line is a partial line. If it is a partial line, it's put into the remnant buffer for the next call to **_transform** to pick up. Finally, it pushes out any non-blank lines. It isn't required for a transform to push any data. Think about implementing a filter as a transform. A filter necessarily drops data out of the stream by not pushing it.

A second method, **_flush**, shows up in this transform due to the way the transform buffers unsent lines in the **remnant** variable. In this way, **_flush** provides an optional way for a transform to empty any data it has buffered during the transformation process when the stream ends. LineTransform uses the **_flush** method to empty any remaining partial lines from the **remnant** variable.

To use the LineTransform, you just add an additional pipe statement to the previous example like this:

```
const BaconReadable =
          require('../BaconReadable');
const LineTransform =
          require('../LineTransform');
const SlackWritable =
          require('../SlackWritable');

const baconReader = new BaconReadable();
const lineTransform = new LineTransform();
// Get a webhook uri at:
const slackWriter = new SlackWritable(
          { webHookUrl: process.env.WEBHOOKURL});

slackWriter.on('finish', () => {
    process.exit(0);
});
```

```
baconReader
        .pipe(lineTransform)
        .pipe(slackWriter);
```

Running this sends bacon ipsum line-by-line to Slack, as shown in **Figure 3,** resulting in 50 messages, one for each paragraph of text emitted by BaconReadable.

Summary

Node streams provide a powerful mechanism to manipulate streaming data. Consider using streams whenever you're reading, writing, or transforming large data sets. Whatever you do, don't cross the streams!

Chris Kinsman

Introduction to the R Programming Language

In my previous article (Nov/Dec 2017 CODE Magazine), I talked about machine learning using Python and the Scikit-learn library. In addition to using Python for data science and machine learning, another language is very popular among data scientist and statisticians, and that's R. R is an open-source programming language and software environment for statistical computing and graphics. R is based on another language, S, created by John Chambers while he was at Bell Labs. The name R was partly due to the names of its two creators, Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand. It was also partly because it was seen as a dialect of the S language.

Wei-Meng Lee
weimenglee@learn2develop.net
www.learn2develop.net
@weimenglee

Wei-Meng Lee is a technologist and founder of Developer Learning Solutions (http://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.

Regardless of the history behind its name, R and its libraries implement a wide variety of statistical techniques, such as linear and nonlinear modeling, classical statistical tests, time-series analysis, classification, clustering, and others. Another core strength of R is graphics, which can produce publication-quality graphs. All of these qualities make R a dream language for statisticians and data scientists.

In this article, I'll start out with an introduction to the R language so that you can get up to speed quickly. In the next article, I'll dive into the various libraries in R that you use for machine learning.

Trying Out R

To try out R, you have a number of options. First, if you followed my previous article on using Python with Scikit-learn, you installed Anaconda (https://www.anaconda.com/download/). (If you didn't already do this, please follow the "Installing Anaconda" sidebar's link.) Although the Anaconda installation only comes with Python support by default, you could easily add R support in Anaconda (https://conda.io/docs/user-guide/tasks/use-r-with-conda.html) by running a simple command in Terminal. To install R in Anaconda, type the following command in Terminal and follow the on-screen instructions:

```
$ conda install r-essentials
```

The above command installs the libraries for R in your Anaconda installation. Once this is done, you can launch Jupyter Notebook. Doing so brings up the development environment using your Web browser:

```
$ jupyter notebook
```

The above command launches the Web browser. To start an R session, click **New > R** (see **Figure 1**).

You should now see the familiar notebook, as shown in **Figure 2**.

Another popular editor for running R code is RStudio (https://www.rstudio.com). I'll be using Jupyter for this article.

Basic Language Syntax

R is a dynamically typed language, meaning that variables need not be pre-declared with a specific data type. Rather, variables take on whatever type is necessary, based on the value assigned to them. The following statements show some examples:

```
num1 <- 5.5
6 -> num2
print(num1)  # 5.5
print(num2)  # 6

num2 = "Two"
print(num2)  # "Two"
```

In R, the assignment operator is **<-** or **->** (although the usual **=** operator is also supported).

To check the data type of variables, use the **typeof()** function:

```
print(typeof(num1))  # "double"
print(typeof(num2))  # "character"
```

You can also perform multiple assignments in a single statement, like this:

```
num2 = 6
num4 <- num3 <- num2
print(num3)                  # 6
print(num4)                  # 6
```

One common misconception when dealing with string variables is to assume that the **length()** function returns the length of the string, as the following example illustrates:

```
str = "This is a string"
print(str)          # "This is a string"
print(length(str))  # 1
```

Interestingly, the **length()** function returns a 1 for the above example. This is because the **length()** function returns the length of vectors (the section on Vectors later in this article covers this more fully). In R, every variable is also of the type **vector**. Think of a vector as an array in a typical conventional programming language. So, in the above example, **length(str)** actually returns the number of items in the **str** vector, which is 1. To get the length of a string variable, use the **nchar()** function, like this:

```
print(nchar(str))    # 16
```

Using Functions in R

In R, you can get more information about a specific function by using the **print()** function. For example:

```
print(exp)
# function (x)  .Primitive("exp")
```

The above code statement shows the **exp()** function, which takes in a single argument and returns a primitive result. Here's another example:

```
print(log)
# function (x, base = exp(1))  .Primitive("log")
```

The **log()** function takes in two arguments. The second argument has a default value of **exp(1)** and the function returns a primitive result.

You can now see how to call the **log()** function using the various combinations of arguments:

```
print(log(10))                # 2.302585
print(log(10, base=exp(1)))   # 2.302585
print(log(10, base=10))       # 1
print(log(10, 10))            # 1
print(log(base=exp(1), x=10)) # 2.302585
print(log(base=exp(1), 10))   # 2.302585
```

Note that you can swap the order of the arguments if you specify the argument names. This is very useful as it makes the function calls much more self-explanatory. They are also some scientific and mathematical functions in R:

```
print(sin(90))                # 0.8939967
print(cos(180))               # -0.5984601
print(tan(270))               # -0.1788391

print(factorial(6))           # 720

print(round(3.14))            # 3
print(round(3.145, 2))        # 3.15
```

Defining Your Own Functions

To define your own function, you can use the **function** keyword and then assign it to a function name, like the following:

```
myFunction <- function(n,m = 6) {
    result = n * m
    result + 5         # OR return (result + 5)
}
```

In the above example, **myFunction** takes two arguments: **n** and **m**. The parameter **m** is known as the default parameter, which has a default value of 6 when you don't supply it when calling the function. Note that the last statement in a function is used as the return value, so essentially the **return** keyword is optional. I prefer to use the **return** keyword, as this makes the function clearer. The following statements show how to call the function, the first with one argument and the second with two arguments:

```
print(myFunction(5))    # 35
print(myFunction(5,7))  # 40
```

Figure 1: Create a new notebook for R.

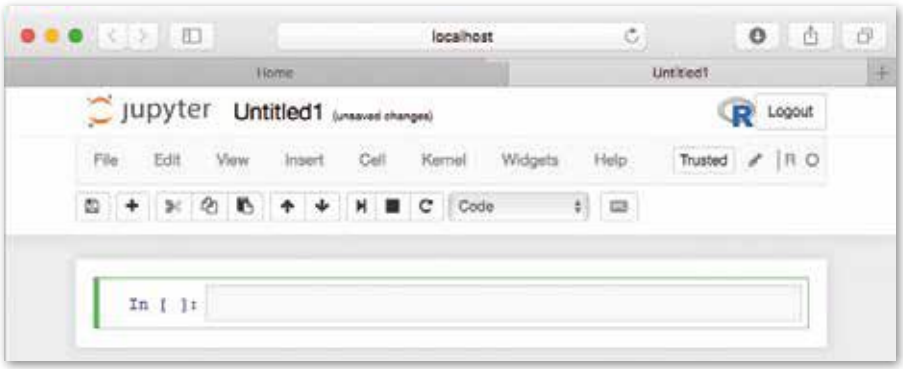


Figure 2: You are now ready for some R action!

Making Decisions

To make decisions, R uses the familiar **if-else** statement construct. The following shows an example of a function that determines if a number is an odd number:

```
isodd <- function(n) {
    if (n %% 2 == 0) {    # %% is modulus
        FALSE
    } else {
        TRUE
    }
}

isodd(46)    # FALSE
isodd(45)    # TRUE
```

If you're one who indulges in terse coding, the above **isodd()** function can be rewritten as a single statement:

```
isodd <- function(n) {(n %% 2 != 0)}
```

R supports the usual arithmetic, relational, and logical operators:

- **+** (addition), - (subtraction), * (multiplication), **/** (division), **^** (power), **%%** (modulo)
- **>** (greater than), **<** (lesser than), **==** (equality), **<=** (lesser or equal to), **>=** (greater than or equal to), **!=** (not equal to)
- **&&** (logical AND), **||** (logical OR), **!** (logical NOT)

The Dot (.) Naming Convention

R uses the dot (.) naming convention in two places. One is as a separator in variables or function names, such as **my.name**, **is.na()**, and **na.omit()**. It's also used as an indicator of the method name of a class/object, e.g., **obj.method()**. This naming convention often trips up programmers coming from other popular OOP languages like C# and Java.

Vectors

As briefly mentioned earlier, everything in R is a vector. Think of a vector as an array of items of the same data type.

```
x <- c(3,4,5,6,7)
print(typeof(x))      # double
print(length(x))      # 5
```

In that snippet, x is a vector comprised of five items of type double. You can append additional items into the vector using the c() function:

```
x <- c(x, 8)          # append another item to x
print(x)              #  3 4 5 6 7 8
```

If you append an item of a different type to the existing vector, R attempts to convert all items in the vector into a common type, as the following example shows:

```
x <- c(x, "9")        # append another item to x
print (x)             #  "3" "4" "5" "6" "7" "8"
                      #  "9"

x <- c(x, TRUE)
print(x)              #  "3"    "4"    "5"
                      #  "6"    "7"    "8"
                      #  "9"    "TRUE"
```

If you want to have a collection of items with *different* types, use a list instead of a vector, like this:

```
y <- list(3,4,5,6,7,"9",TRUE)
```

Vector Functions

In R, there are a number of vector functions that make manipulating numbers easy. Here's an example:

```
nums <- c(12,34,56,9,45,67,90,11,2,45)
print(min(nums))     # 2
print(max(nums))     # 90
print(mean(nums))    # 37.1
print(median(nums))  # 39.5
print(sd(nums))      # 28.90002
print(sort(nums))    # 2  9 11 12 34 45 45 56
                     # 67 90
print(sum(nums))     # 371
print(unique(nums))  # 12 34 56  9 45 67 90
                     # 11  2
```

As you can see, given a vector containing numbers, it's very easy to get information from the vector using the various vector functions such as **min()**, **max()**, etc. In particular for data science, you can use the **summary()** function to get a summary of the numbers contained in the vector:

```
print(summary(nums))

#    Min. 1st Qu.  Median    Mean 3rd Qu.
#    Max.
#    2.00   11.25   39.50   37.10   53.25
#    90.00
```

Dealing with NAs

A lot of times, you load data from files, in particular, CSV or tab-separated files. These data files may contain "holes" in them, meaning missing data for some rows and columns. When loading missing values from a CSV file, R automatically replaces them with NAs (for Not Available). One good use of the **summary()** function is to show the number of NAs in a vector so that you can decide if you need to replace or omit them from the vector before you do any further processing. The following code example shows the **summary()** function displaying the number of NAs in the vector:

```
nums <- c(12,34,56,9,45,67,90,11,2,45, NA)
print(summary(nums))
#    Min. 1st Qu.  Median    Mean 3rd Qu.
#    Max.          NA's
#    2.00   11.25   39.50   37.10   53.25
#   90.00   1
```

To omit the NAs in your vector, use the **na.omit()** function:

```
print(sum(na.omit(nums)))  # omit the NAs in
                           # the vector and
                           # sum up the rest
```

If you want to know whether each element in the vector is a NA, use the **is.na()** function:

```
print(is.na(nums))         # FALSE FALSE FALSE
                           # FALSE FALSE FALSE
                           # FALSE FALSE FALSE
                           # FALSE  TRUE
```

To extract all the numbers in a vector that isn't an NA, use the **subset()** function:

```
nums <- subset(nums, is.na(nums) == FALSE)
print(nums)    # 12 34 56  9 45 67 90 11  2 45
```

Sequencing

When performing data science operations, you often need to generate a sequence of numbers. Instead of creating a vector of numbers manually, it would be easier to be able to generate the sequence automatically. The following code snippet generates a sequence from 1 to 3:

```
i <- 1:3      # generate a sequence from
              # 1 to 3
print(i)      # 1 2 3
```

You can also generate a sequence in the reverse order:

```
j <- 5:1
print(j)      # 5 4 3 2 1
```

You can also use the **seq()** function to generate a sequence of numbers:

```
y <- seq(9)
print(y)      # 1 2 3 4 5 6 7 8 9
```

The **seq()** function is useful when you want to specify an increment for the sequence, like the following:

```
y <- seq(from=2, to=9, by=2)
print(y)               # 2 4 6 8
```

You can also specify the length of the sequence as well as the starting and ending number and the **seq()** function automatically divides the numbers equally:

```
y <- seq(from=1, to=1.9, length=20)
print(y)
#  [1] 1.000000 1.047368 1.094737 1.142105
#      1.189474 1.236842 1.284211 1.331579
#  [9] 1.378947 1.426316 1.473684 1.521053
#      1.568421 1.615789 1.663158 1.710526
# [17] 1.757895 1.805263 1.852632 1.900000
```

If you want to generate a sequence of identical numbers, use the **rep()** (for repeat) function:

```
zeros <- rep(0, time=20)
print(zeros)              # 0 0 0 0 0 0 0 0 0 0
                          # 0 0 0 0 0 0 0 0 0 0
```

The **rep()** function can also be used to repeat a vector, as the following demonstrates:

```
cars <- c("Suzuki", "Toyota", "Mercedes")
manycars <- rep(cars, each=2)
print(manycars)
# "Suzuki"   "Suzuki"   "Toyota"   "Toyota"
# "Mercedes" "Mercedes"
```

Vector Indexing

You can use a sequence as an index into a vector to retrieve the items you want, as the following shows:

```
x <- c(3,4,5,6,7)
i <- 1:3
print(x[i])             # 3 4 5
```

To omit an item in a vector, specify the index of the item to omit and prefix it with a negative sign, like this:

```
print(x[-1])          # 4 5 6 7
                      # excludes the first item

print(x[-2])          # 3 5 6 7
                      # excludes the second item

print(x[-length(x)])  # 3 4 5 6
                      # excludes the last item
```

To get the last n items from a vector, use the **tail()** function, like this:

```
print(tail(x,1))        # 7
print(tail(x,2))        # 6 7
```

You can also specify a range of items to extract from a vector, like this:

```
print(x[-2:-3])      # 3 6 7
                     # excludes second through
                     # third items

print(x[1:length(x)-1]) # 3 4 5 6
                     # excludes the last item
```

You can also specify conditions, like this:

```
print(x[x > 5])              #  6 7
print(x[x %% 2 == 0])        #  4 6
```

Looping

R supports looping constructs that are commonly found in other programming languages. The following example shows the **for** loop in action:

```
# function to print the first n numbers of the
# fibonacci numbers
fib <- function(n) {
    x <- c(1,1)
    for (i in 2:(n-1)) {
        x <- c(x, sum(tail(x,2)))
    }
    return (x)
}

print(fib(8))      # 1  1  2  3  5  8 13 21
```

You can also use the **while** loop in R; in the following example, I used it to create a Fibonacci sequence up to the number specified:

```
# function to print the fibonacci sequence until
# the n numbers specified
fib <- function(n) {
    x <- c(1,1)
    while (tail(x,1) < n) {
        x <- c(x, sum(tail(x,2)))
    }
    return (x)
}

print(fib(13))               # 1  1  2  3  5  8 13
```

Data Frames

A lot of times, data is represented in tabular format. This is something that R excels in. Date Frames are extremely useful in data science operations as data is often stored in CSV files or Excel spreadsheets. Loading the data into data frames allows you to manipulate the data using rows and columns.

Creating a Data Frame from Vectors

The following code snippet shows how a data frame (think of it as a table) is created from three vectors:

```
# column 1
c1 = c(2, 3, 5)

# column 2
c2 = c("aaa", "bbb", "ccc")

# column 3
c3 = c(TRUE, FALSE, TRUE)

# create a data frame using the 3 columns
df = data.frame(c1,c2,c3)

print(df)
```

Printing the data frame produces the following output:

```
  c1  c2    c3
1  2 aaa  TRUE
```

Installing Anaconda

Anaconda is a package that contains the Python interpreter as well as a suite of libraries and packages for creating data science and machine learning projects. You can download Anaconda from https://www.anaconda.com/download/. You can download Anaconda for Windows, Mac OS, and Linux. Once Anaconda is installed, you can type a command in the command window to install the support for R. For Windows users, launch the Anaconda Prompt instead of the regular Command window. Just type the word "Anaconda" in the Search box and you should see the Anaconda Prompt application. The Anaconda Prompt is just like a regular Command window, except that it has the paths to the relevant utilities in Anaconda all set up for you.

```
2  3 bbb FALSE
3  5 ccc  TRUE
```

Changing the Column Names of a Data Frame

Note that the column names take on the names of the three vectors by default. You can change this by specifying the column name explicitly:

```
df = data.frame(col1 = c1, col2 = c2, col3 = c3)
print(df)
```

The above changes produce the following output with the new column names:

```
  col1 col2 col3
1    2  aaa TRUE
2    3  bbb FALSE
3    5  ccc TRUE
```

Extracting Columns

To print out a specific column, use the index of the column (remember, index in R starts with 1, not 0):

```
print(df[2])    # second column
```

The above prints out this:

```
  col2
1  aaa
2  bbb
3  ccc
```

You can also print out a column using its column name, like this:

```
print(df["col3"])
'
   col3
1  TRUE
2 FALSE
3  TRUE
'
```

Note that the preceding prints out a data frame containing only the third column. If you want the values of the third column as a vector, you can use the following syntax:

```
print(df$col3)  # same as print(df[,"col3"])
'
[1]  TRUE FALSE  TRUE
'
```

Extracting Rows

To print out a specific row in a data frame, specify its row index:

```
print(df[2,])
'
  col1 col2  col3
2    3  bbb FALSE
'
```

To print out a specific item in a data frame, specify its index and column number:

```
print(df[2,3])
'
[1] FALSE
'
```

Transposing a Data Frame

You can also transpose a data frame using the **t()** function:

```
print(t(df))
'
     [,1]    [,2]    [,3]
col1 "2"     "3"     "5"
col2 "aaa"   "bbb"   "ccc"
col3 " TRUE" "FALSE" " TRUE"
'
```

The **t()** function converts rows to columns and columns to rows.

Subsetting Data Frames

To create a subset of a data frame based on certain criteria, use the **subset()** function:

```
true_col = subset(df, col3 == TRUE)
print(true_col)
'
  col1 col2 col3
1    2  aaa TRUE
3    5  ccc TRUE
'
```

The previous snippet retrieves a subset of the **df** data frame based on the value of the "**col3**" column. The statements in this next snippet filters the result based on column names:

```
true_col = subset(df, col3 == TRUE,
                  select = c(col1, col2))
print(true_col)
'
  col1 col2
1    2  aaa
3    5  ccc
'
```

Creating a Data Frame from Files

For data science work, most of the time you create a data frame directly from a file, such as a CSV, or tab-separated file. Suppose you have the following content saved as a file named **fruits.csv**:

```
orange,pineapple,durian
2,3,4
4,5,2
5,3,1
3,2,5
6,8,10
```

The following code snippet reads the above CSV file into a R data frame and prints it out:

```
fruits <- read.csv(file = "fruits.csv")
print(fruits)
'
  orange pineapple durian
1      2         3      4
2      4         5      2
3      5         3      1
```

The c() Function

The c() function is known as the concatenation function. It allows you to combine items into a vector. It also allows you to append items into an existing vector.

```
4      3         2      5
5      6         8     10
'
```

Sometimes the CSV files might not be located locally but resides on a Web server. In this case, you can download it as a file before reading it into a data frame:

```
download.file(url =
    "http://bit.ly/2iJjdpb",
    destfile = "crimerecords.csv")
crimerecords <-
    read.csv(file = "crimerecords.csv")
```

The preceding prints out the output, as shown in **Figure 3**. The **head()** function returns the first *n* rows of the data frame, which by default is the first six rows. Likewise, if you want to print the last *n* rows of the data frame, use the **tail()** function.

Plotting Charts Using Data Frame

One of the key features of R is its strong graphic capabilities. Using R, you can directly plot graphs and charts. In the following sections, I'll discuss how to plot some interesting charts in R.

Plotting Bar Charts

Suppose you have a CSV file named **public_transport.csv** containing the following content:

```
Gender,Age,Mode,Times
male,25,bus,20
female,28,train,10
male,35,bicycle,20
male,23,bus,7
female,43,bus,24
female,19,train,16
male,41,bus,28
female,12,bicycle,10
male,32,bus,19
female,29,train,19
female,11,bus,7
male,22,train,8
female,26,train,23
female,27,train,31
male,37,train,32
male,31,train,22
```

That CSV file contains a listing of commuters and their age, their mode of transport and how many times they used the specified mode of transport every month. The following code snippet first loads the CSV file into a data frame:

```
freq <- read.csv(file = "public_transport.csv")
# count the total occurences of each "Mode" of
# transportation
```

Suppose you want to know the number of people using bicycles, buses, and trains. To do this, you can use the **table()** function, which will help you tabulate the frequencies of each occurrence of transport mode:

```
freqMode <- table(freq$Mode)
print(freqMode)
'
```

```
    cdatetime              address district       beat grid
1 1/1/06 0:00  3108 OCCIDENTAL DR        3 3C          1115
2 1/1/06 0:00 2082 EXPEDITION WAY        5 5A          1512
3 1/1/06 0:00           4 PALEN CT       2 2A           212
4 1/1/06 0:00       22 BECKFORD CT       6 6C          1443
5 1/1/06 0:00     3421 AUBURN BLVD       2 2A           508
6 1/1/06 0:00   5301 BONNIEMAE WAY       6 6B          1084
                     crimedescr ucr_ncic_code latitude longitude
1 10851(A)VC TAKE VEH W/O OWNER          2404 38.55042 -121.3914
2    459 PC  BURGLARY RESIDENCE          2204 38.47350 -121.4902
3 10851(A)VC TAKE VEH W/O OWNER          2404 38.65785 -121.4621
4  476 PC PASS FICTICIOUS CHECK          2501 38.50677 -121.4270
5  459 PC  BURGLARY-UNSPECIFIED          2299 38.63745 -121.3846
6 530.5 PC USE PERSONAL ID INFO          2604 38.52698 -121.4513
```

Figure 3: The output of the data frame loaded from a URL

```
bicycle     bus   train
      2       6       8
'
```

Using the result returned by the **table()** function, you can plot a bar chart using the **barplot()** function:

```
barplot(freqMode,
        main="Main Mode of Transportation",
        xlab="Modes",
        ylab="Total")
```

The **main** argument specifies the title of the chart, while the **xlab** and **ylab** arguments specify the x-axis and y-axis labels respectively. The bar chart created is shown in **Figure 4**.

You can also alter the density of the bars by specifying the **density** argument:

```
barplot(freqMode,
        main="Main Mode of Transportation",
        xlab="Modes",
        ylab="Total",
        border="blue",
        density=c(10,20,30,40,50))
```

The density argument is a vector giving the density of shading lines, in lines per inch, for the bars or bar components. The bar chart now looks like **Figure 5**.

Besides loading a data frame from a CSV file, you can also load it from a tab-separated file, such as the following example of a file named **fruits.txt** that contains the sales of the various fruits for each month from January to May:

```
orange pineapple    durian
2    3    4
4    5    2
5    3    1
3    2    5
6    8    10
```

The following code snippet loads the content of the file into a data frame:

```
fruits_data <- read.table("fruits.txt",
                          header=T,
                          sep="\t")
```

Index in R

Vector index in R starts with 1, not 0 as in most programming languages, like Java and C#.

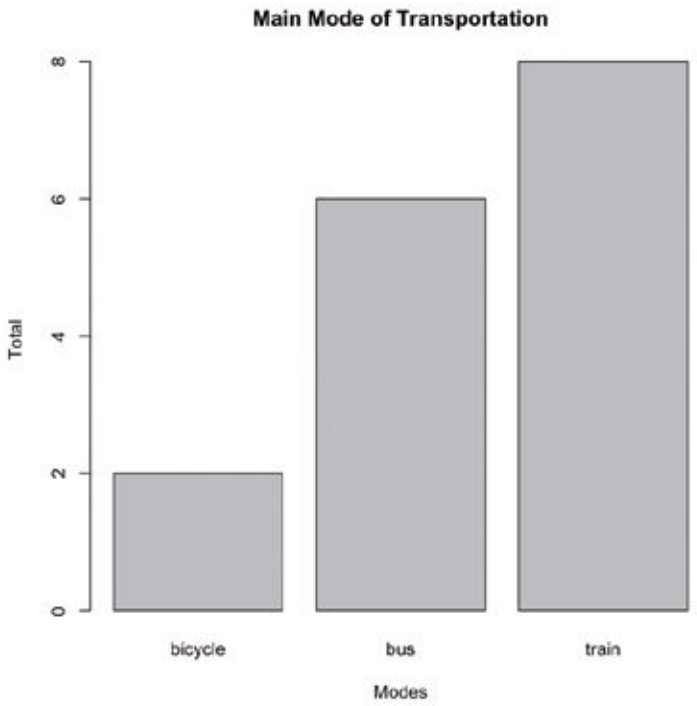


Figure 4: Plot a bar chart showing the use of the various modes of transport.

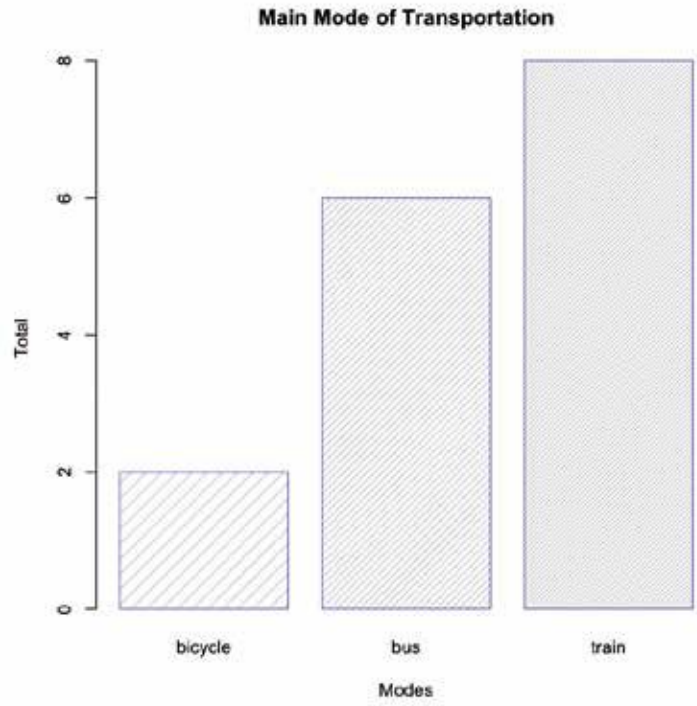


Figure 5: Change the shading of the bars

Next, plot a bar chart showing the sales of oranges for each of the five months:

```
barplot(fruits_data$orange,
        main="Sales of Oranges",
        xlab="Months",
        ylab="Total",
        names.arg=c("Jan","Feb","Mar","Apr","May"))
```

The **names.arg** argument is a vector of names to be plotted below each bar or group of bars. The bar chart is shown in **Figure 6**.

You can change the colors of the bars (see **Figure 7**) by using the **col** argument:

```
fruits_data <- read.table("fruits.txt",
                          header=T,
                          sep="\t")
barplot(fruits_data$orange,
        main="Sales of Oranges",
        xlab="Months",
        ylab="Total",
        names.arg=c("Jan","Feb","Mar",
                    "Apr","May"),
        col=rainbow(5))
```

The col argument is a vector of colors for the bars or bar components. You can use the **rainbow()** function to generate a set of colors. For example, the **rainbow(5)** statements generates a vector containing the following five elements: "#FF0000FF", "#CCFF00FF", "#00FF66FF", "#0066FFFF", and "#CC00FFFF".

What about displaying the sales of the various fruits for each month? You can first convert the data frame into a matrix using the **as.matrix()** function:

```
print(as.matrix(fruits_data))
'
     orange pineapple durian
[1,]      2         3      4
[2,]      4         5      2
[3,]      5         3      1
[4,]      3         2      5
[5,]      6         8     10
'
```

And then plot the bar chart using the matrix:

```
fruits_data <- read.table("fruits.txt",
                          header=T,
                          sep="\t")
barplot(as.matrix(fruits_data),
        main="Sales of Fruits from Jan to May",
        xlab="Months",
        ylab="Total",
        beside=TRUE,
        col=rainbow(5))
```

The **beside** argument specifies whether the columns of height are portrayed as stacked bars (FALSE), or the columns are portrayed as juxtaposed bars (TRUE). **Figure 8** shows the bar chart with the **beside** argument set to **TRUE**.

Figure 9 shows the bar chart with the **beside** argument set to **FALSE**.

You can display a legend on your chart using the **legend()** function:

```
fruits_data <- read.table("fruits.txt",
                          header=T,
                          sep="\t")
barplot(as.matrix(fruits_data),
```

```
    main="Sales of Fruits from Jan to May",
    xlab="Fruits",
    ylab="Total",
    beside=TRUE,
    col=rainbow(5))

legend("topleft",
       c("Jan","Feb","Mar","Apr","May"),
       cex=1.6,
       bty="n",
       fill=rainbow(5));
```

The **cex** argument specifies the size of the text to be used for the legend. The **bty** argument takes either "n" or "o". Setting to "o" draws a rectangle around the legend and setting it to "n" means no rectangle is drawn.

Figure 10 shows the chart with the legend.

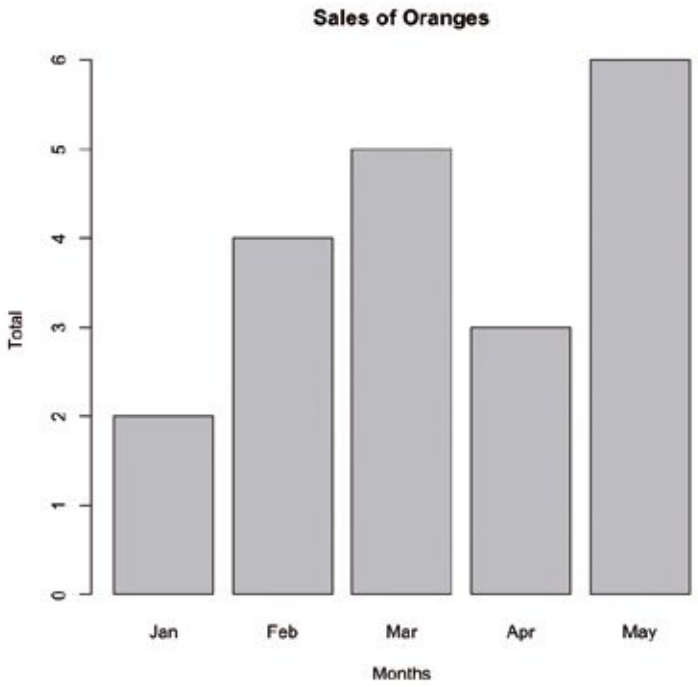


Figure 6: Plot the sales of oranges for each month

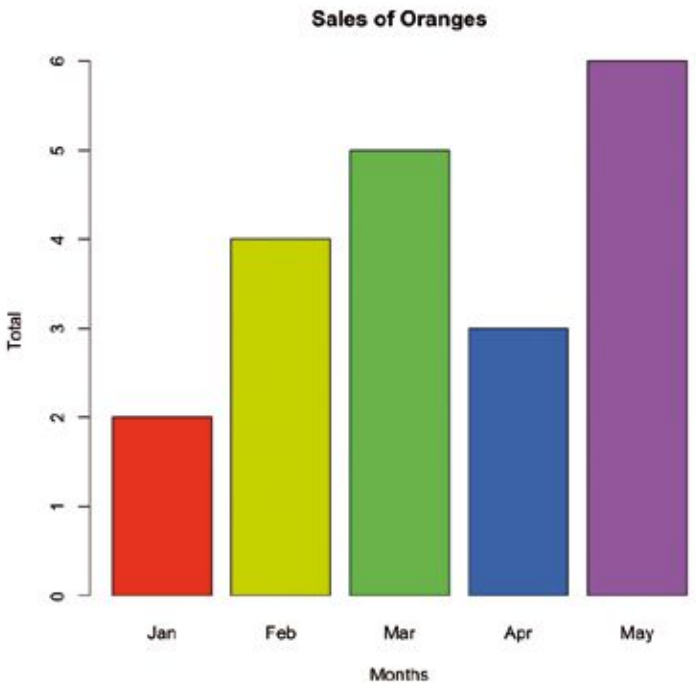


Figure 7: Change the colors of the bars.

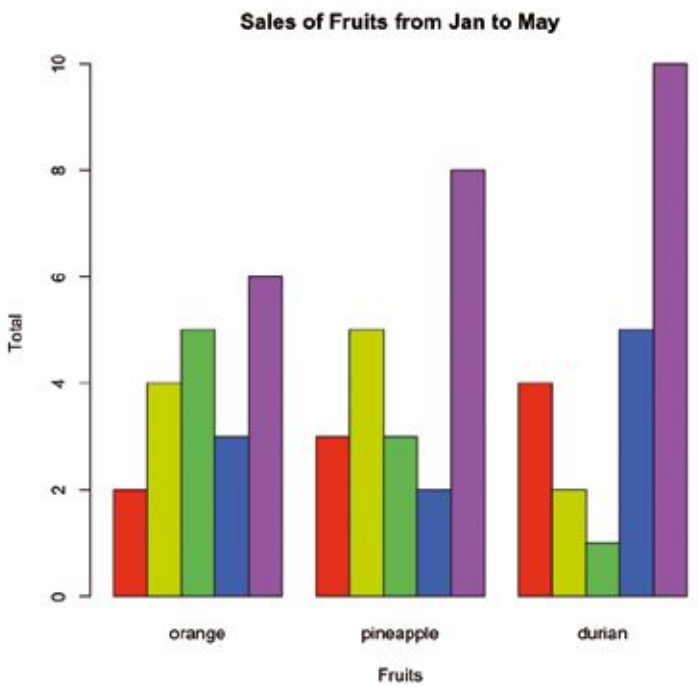


Figure 8: Plot the sales of all the fruits for each month.

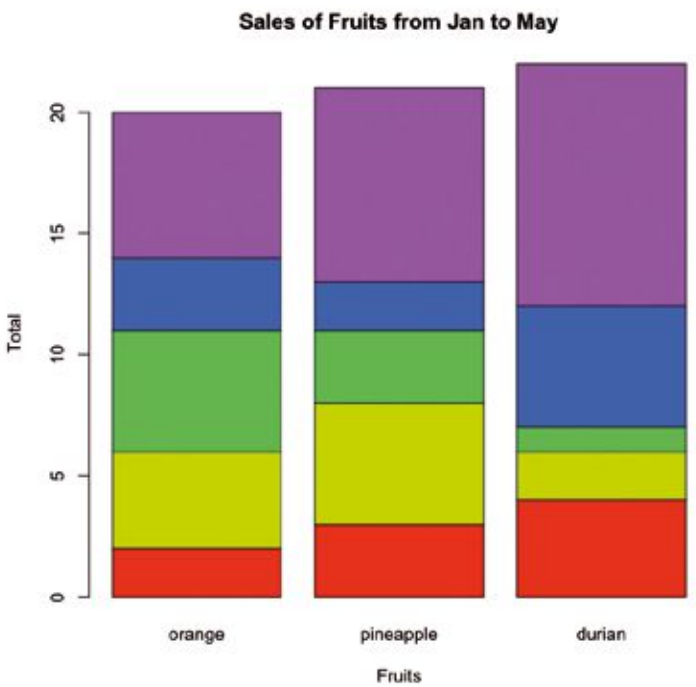


Figure 9: Stack the bars of each month for each of the fruits.

Plotting Histograms

A histogram is another type of chart that's very useful for showing the distribution of numerical data. Using the transportation CSV file that I discussed earlier, you can plot the distribution of the commuters' age using a histogram, as shown in the following code snippet:

```
freq <- read.csv(file = "public_transport.csv")
histogram = hist(freq$Age,
           main ="Distribution of Age groups",
           xlab = "Age Range",
           ylab = "Total")
```

Figure 11 shows the histogram showing the distribution of the ages of the commuters.

Observe that R automatically breaks the data up into intervals. You can verify this by printing the **breaks** property:

```
print(histogram$breaks)
'
[1] 10 15 20 25 30 35 40 45
'
```

Sometimes you want to have more control over the breaks, and you can indeed do so via using the **breaks** argument by passing it a sequence, like this:

```
histogram = hist(freq$Age,
           main ="Distribution of Age groups",
           xlab = "Age Range",
           ylab = "Total",
           breaks = seq(0,50, by=10))
```

The updated histogram looks like **Figure 12.**

Plotting Scatter Plots

Scatter plots are useful for plotting data points on a horizontal and a vertical axis when attempting to show how much one variable is affected by another. Consider the following example CSV file, named **rainfall.csv,** containing the yield of a particular crop and the associated rainfall and average temperature for a particular year.

```
year,yield,rainfall,temperature
1963,60,8,56
1964,50,10,47
1965,70,11,53
1966,70,10,53
1967,80,9,56
1968,50,9,47
1969,60,12,44
1970,40,11,44
```

You could plot a scatter plot using the **plot()** function:

```
rainfall <- read.csv(file = "rainfall.csv")
plot(rainfall[1:4])
```

Figure 13 shows the scatter plot.

Using the scatter plot, you can compare the relationships among the various factors, such as year, yield, rainfall, and temperature. **Figure 14** shows how to read the scatter plot.

Plotting Pie Charts

A pie chart displays a circle divided into slices to illustrate numerical proportion. In R, you can display a pie chart using the **pie()** function. Consider the following code snippet, which has a vector containing the market share of operating systems:

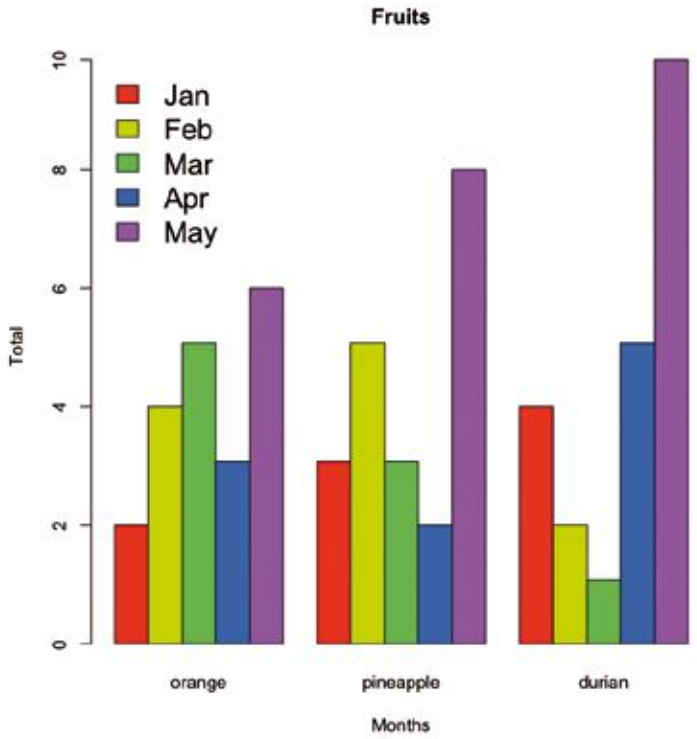


Figure 10: Display a legend with the bar chart

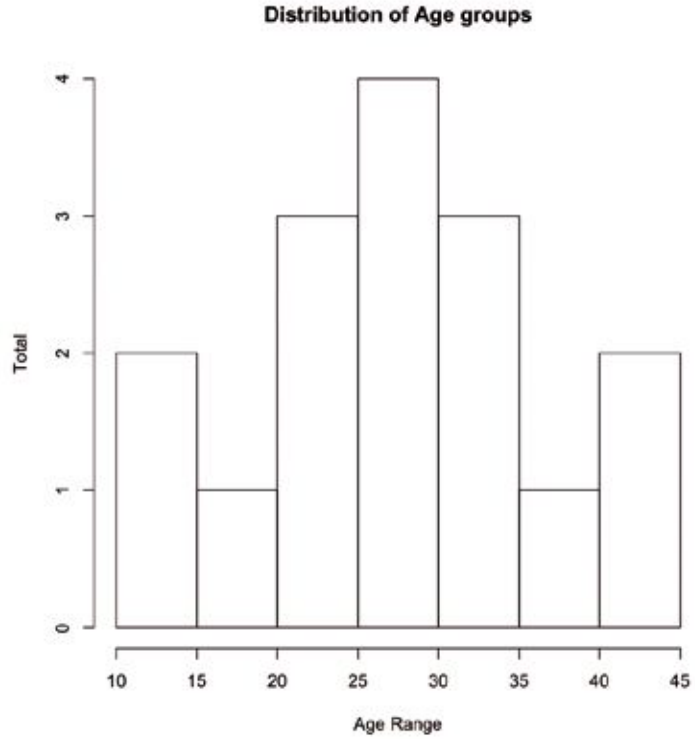


Figure 11: Use a histogram to show the distribution of age among the commuters

```
os <- c(63.99, 32.03, 1.48, 1.14, 0.84, 0.51)
pie(os)
```

The pie chart created is shown in **Figure 15**.

You could supply additional information to make the pie chart more descriptive by adding labels:

```
pie(os,
    main="Mobile OS",
    col=rainbow(length(os)),
    labels=c("Android","iOS","Windows Phone",
             "Java ME","Symbian", "Others"))
```

Figure 16 shows the pie chart that is color-coded, with labels representing each slice of the pie.

As you saw earlier, you can generate a vector of color using the **rainbow()** function. In addition to this function, you can use the various other color palettes in R:

- heat.colors(n)
- terrain.colors(n)
- topo.colors(n)
- cm.colors(n)

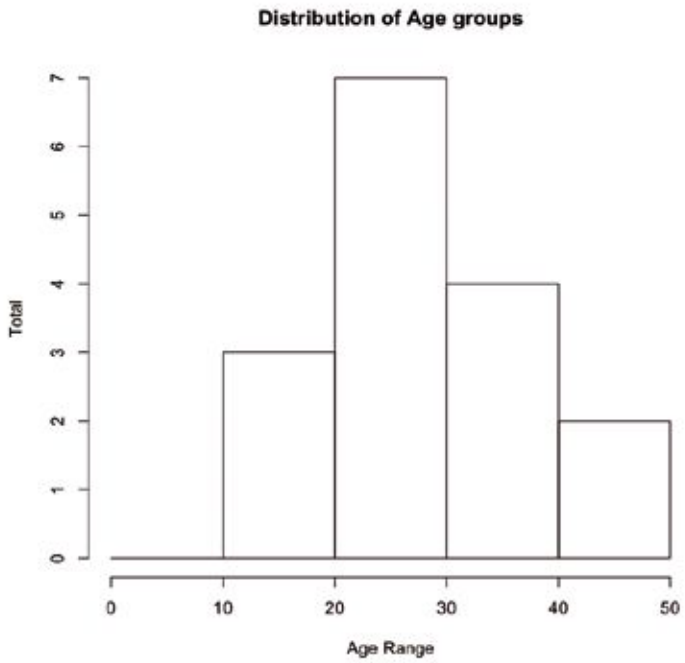


Figure 12: Alter the breaks in the histogram

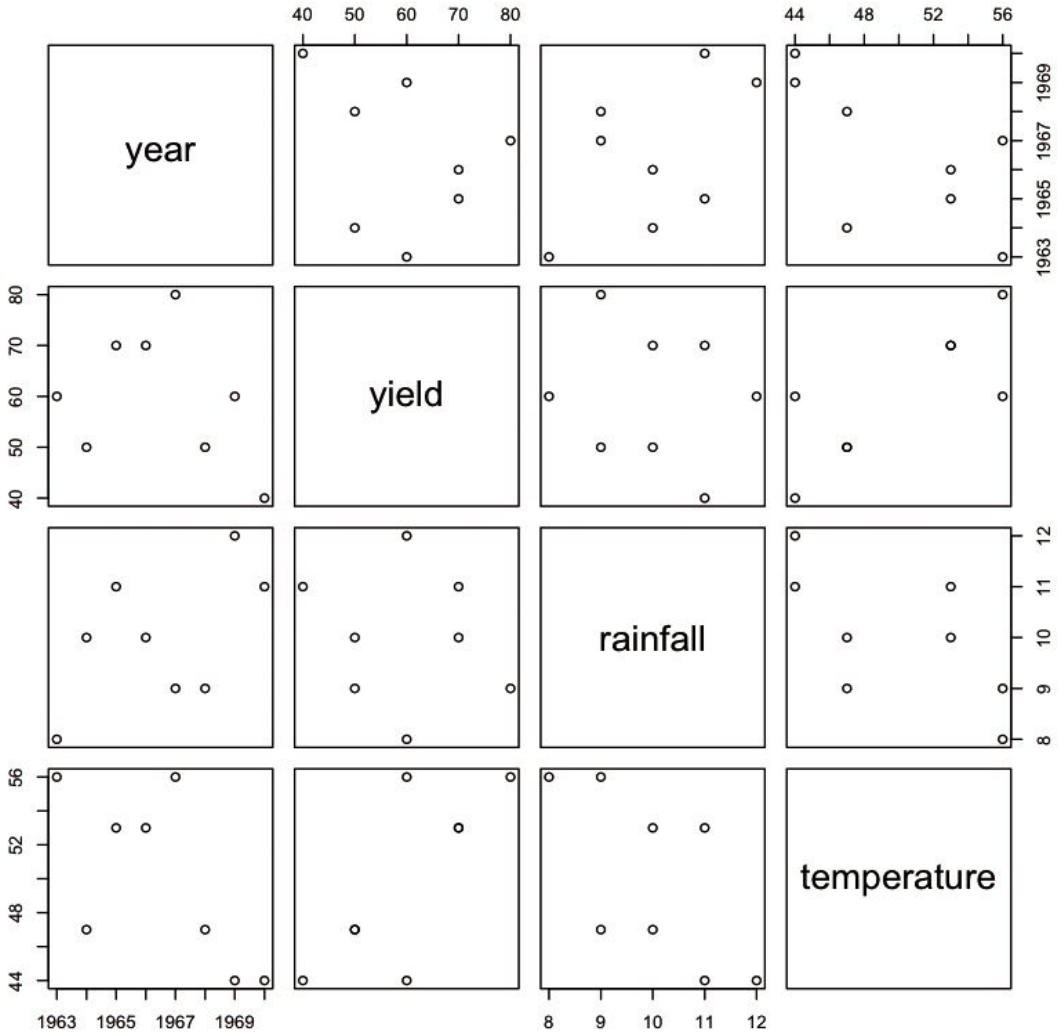


Figure 13: Show the relationships among all the variables using a scatterplot.

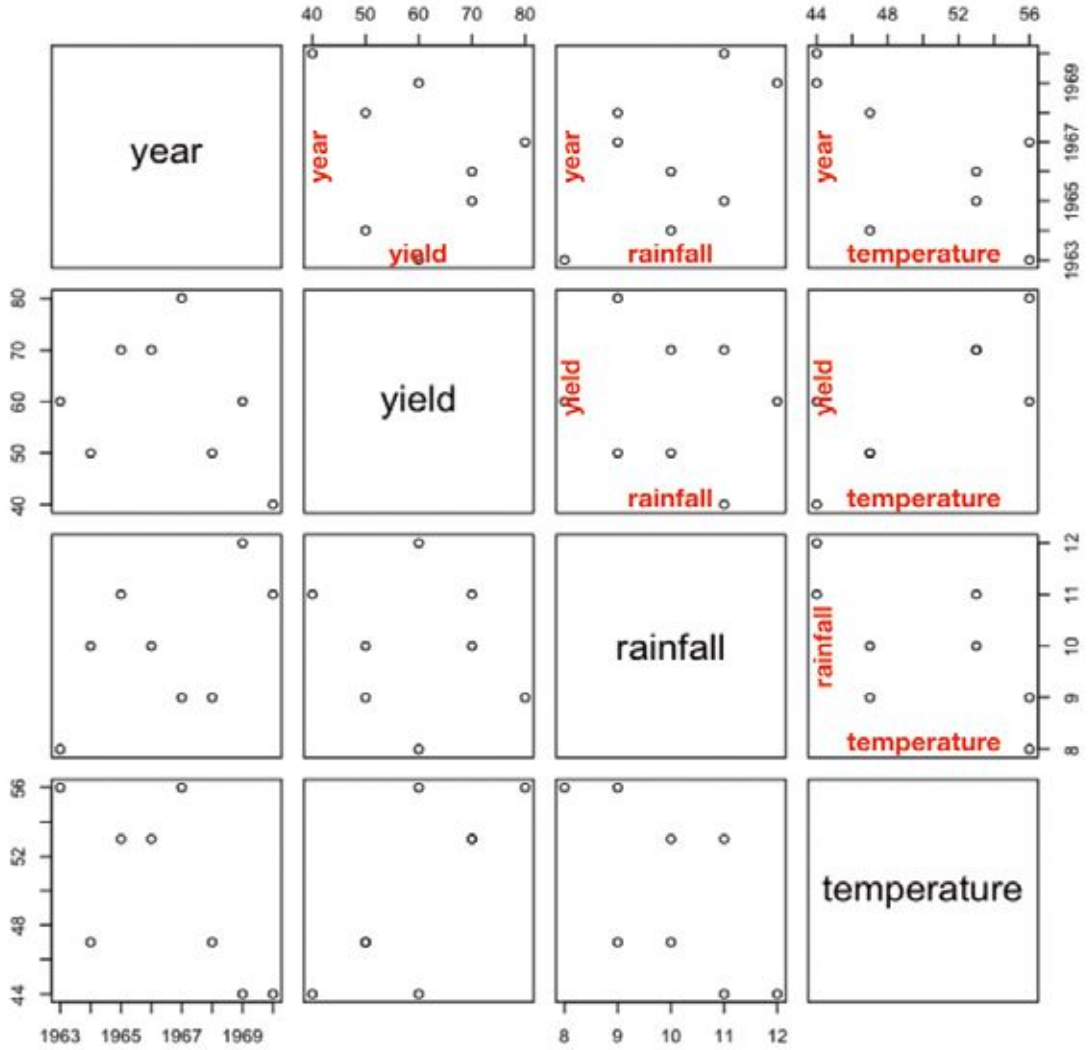


Figure 14: How to read a scatterplot

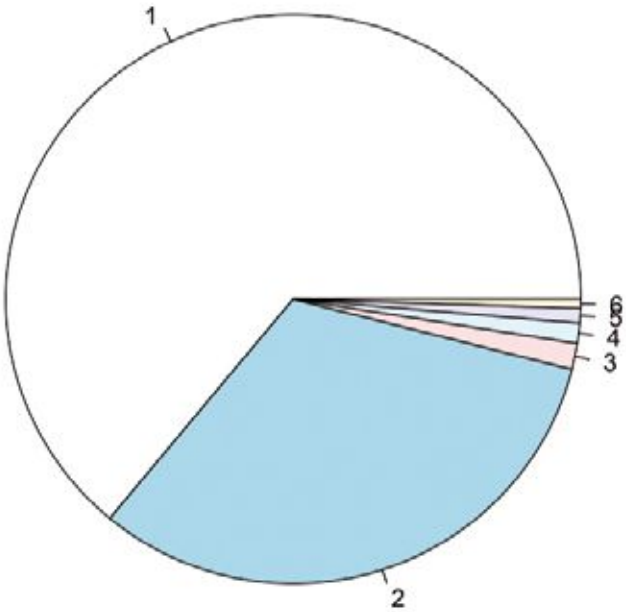


Figure 15: Displaying a pie chart

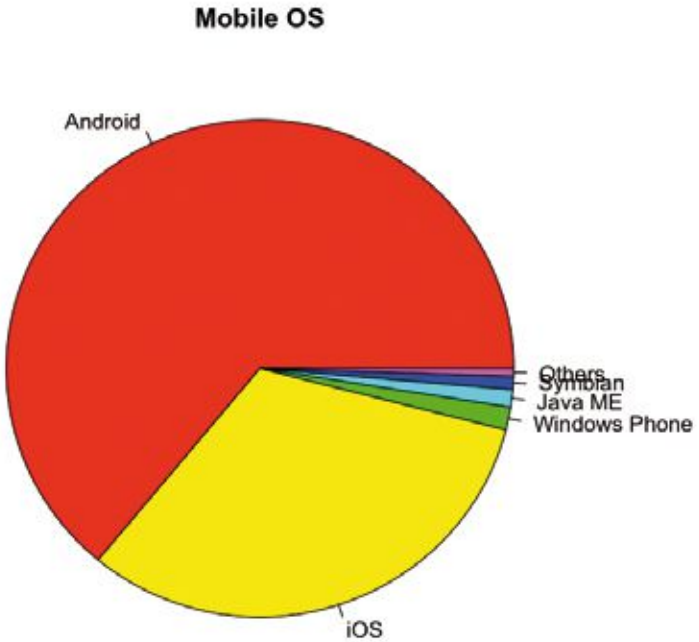


Figure 16: Color-code the pie chart

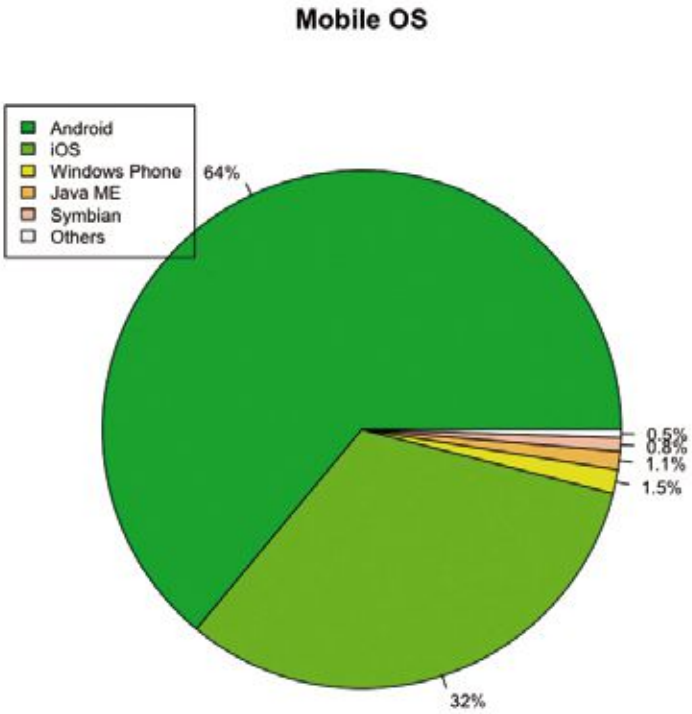


Figure 17: Show a legend for the pie chart and percentages for each slice

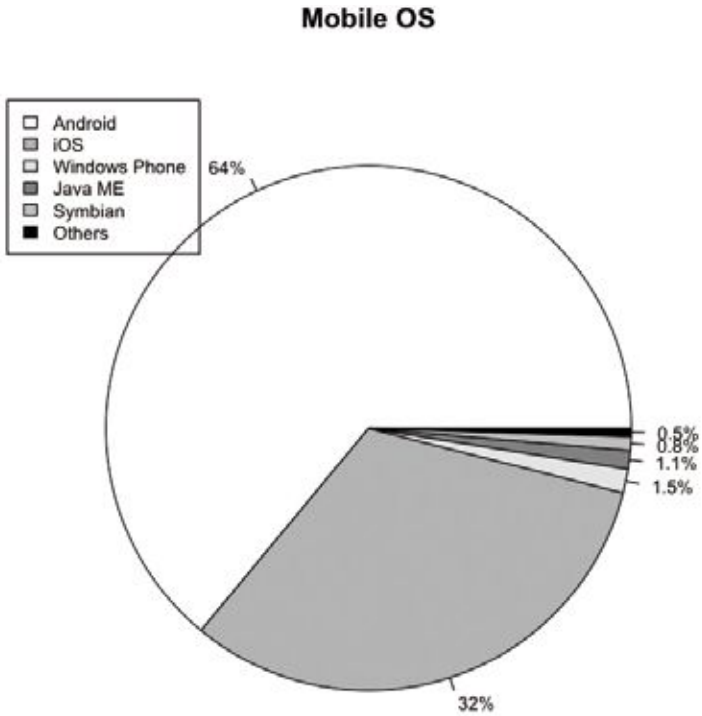


Figure 18: Display the slices using shades of grey

The following code snippet shows a more detailed pie chart (see **Figure 17**) displaying the percentage of each slice as well as displaying a legend:

```
colors <- terrain.colors(6)

# calculate the percentage
os_labels <- round(os/sum(os) * 100, 1)

# concat a % after each value
os_labels <- paste(os_labels, "%", sep="")

pie(os,
    main="Mobile OS",
    col=colors,
    labels=os_labels, cex=0.8)

legend(-1.1,
       1,
       c("Android","iOS","Windows Phone",
         "Java ME","Symbian", "Others"),
       cex=0.8,
       fill=colors)
```

Besides the palette of colors, you can also create your own sets of grey tones:

```
colors <- c("white","grey70","grey90",
            "grey50","grey75","black")
```

Figure 18 shows the pie chart using different shades of grey.

Summary

In this article, you had a whirlwind tour of R. Although this isn't an attempt to teach you everything about R, I do believe getting acquainted with the language can make you ready to embrace machine learning. In addition to the language basics, you also learned how to visualize data by using the various functions in R to plot charts, such as pie charts, bar charts, histograms, and scatterplots.

Wei-Meng Lee

Azure Skyline: Terms

Azure Resource Groups, App Service Plans, and SQL Elastic Pools. These terms aren't very well understood by most developers, but it turns out that they're easy-to-learn ideas and knowing how to use them can save you both time and money.

Mike Yeager
www.internet.com

Mike is the CEO of EPS's Houston office and a skilled .NET developer. Mike excels at evaluating business requirements and turning them into results from development teams. He's been the Project Lead on many projects at EPS and promotes the use of modern best practices, such as the Agile development paradigm, use of design patterns, and test-drive and test-first development. Before coming to EPS, Mike was a business owner developing a high-profile software business in the leisure industry. He grew the business from two employees to over 30 before selling the company and looking for new challenges. Implementation experience includes .NET, SQL Server, Windows Azure, Microsoft Surface, and Visual FoxPro.

Resource Group

Every time you add a resource to your Azure account, whether it's a virtual machine, a database, or any of the other hundreds of resources Azure offers, you have to assign the resource to a Resource Group, but what is a Resource Group? A Resource Group is nothing more than a logical grouping of Azure resources. The Resource Group itself doesn't use any resources and you don't get billed for it. It's just a list. The Resource Group doesn't have to be in the same location as any of the resources in it. Technically, it doesn't matter which resource group you put resources in and most Azure demos tell you to just make one up and leave it at that. So what good are Resource Groups? What's the RIGHT way to use Resource Groups?

You put all of the resources for each project in the same Resource Group. A standard here at CODE Magazine/EPS is to name the Resource Group after the project. There are a couple of exceptions to this mentioned below, but this is a great rule of thumb.

> Naming the Resource group after the project is a good rule of thumb.

It's easy to see all of the parts of a project on one screen and to see what you're being billed per project. What if you have resources that aren't in the right Resource Group? You can move resources from one Resource Group to another using the Azure portal. You can even move resources to a Resource Group in another subscription, as long as you can see both subscriptions in the portal under the same login. This used to be impossible. Later it was possible but a gigantic pain. Now it's pretty easy, as shown in **Figure 1**.

App Service Plan

You can think of an App Service Plan as a logical IIS Server instance. Treat it just as you would an IIS Server. You can run a whole bunch of App Services (websites, Web jobs, Web API services) on the same App Service Plan. You get charged by the App Service Plan, not by the App Service. Changing the pricing tier of an App Service Plan changes every App Service on that plan. This is one place where you might deviate from the rule of-thumb for Resource Groups. You may want to share an App Service Plan across several projects versus paying for an App Service Plan for each project. Of course, the lowest pricing tier is free, and it doesn't cost any more to have 30 free App Service Plans than it does to share just one. But, if you need or want better performance or features than the free tier offers, you can share the cost of a single App Service Plan across many App Services.

SQL Elastic Pool

A SQL Elastic Pool is a special resource attached to a regular Azure SQL Server instance. Like an Azure SQL Server, the SQL Elastic Pool is a logical resource, not a physical one. It's really nothing more than a list of databases and settings that can have SQL databases assigned to them. The key difference is that on an Azure SQL Server, each database is charged and scaled separately. An Elastic Pool server shares resources (DTUs) and costs across all databases on the server. If you have four or five Basic SQL Databases at $5/month, this won't help you much, as the minimum pool size is 50 DTUs. If you're using 50 or more DTUs, you might be able to save some money with an Elastic Pool.

COME Magazine/EPS recently had a client running three PRS1 instances, which is 125 DTUs each. We saved them money by creating an Elastic Pool at PRS2, which is 250 DTUs. All three databases share the 250 DTUs and pricing is similar to a single 250 DTU server. Because it would be extremely rare for all three databases to require high DTU

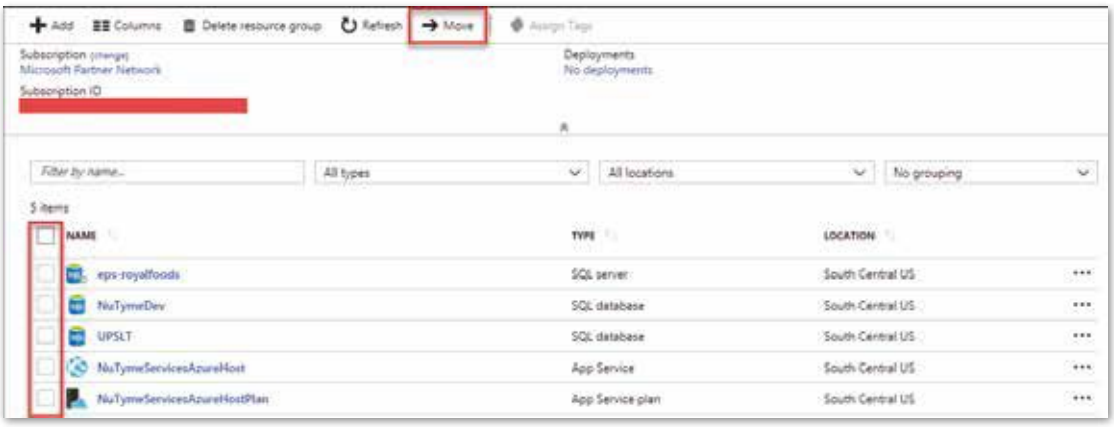


Figure 1: You can see all resources in a Resource Group and move them to another Resource Group.

rates at the same time, the perception of the client's users is that each database is running about twice as fast as it used to while costing 1/3 less than it used to.

Summary

A few minutes familiarizing yourself with some common Azure terminology can make life a bit easier and even save you some money. Those included here are simplified explanations and not 100% complete, but they're plenty to help you be more productive. if you're the curious type or you wind up using these ideas extensively, I recommend reading the exceedingly long-winded, but complete and accurate versions in the documentation. You'll find even more useful information there.

Mike Yeager

(Continued from 74)

comes along that provides even the slightest hint of a distraction, you're on it.

Why should this be any different if you're the boss? Particularly in those situations where you weren't promoted up out of the team, they're going to have the same skepticism about "the new boss" as you would. Getting your team to commit to you as a manager takes time and trust.

The Myth of the Trains

Also known as, "My job as a manager is to make sure the trains continue to run on time—that the operation (development, QA, or whatever it is) runs smoothly." And the corollary to that myth? That the fires that emerge are yours to fight. Any fire that appears is on you to deal with—you're the manager, right? As the US President famously put it, "The buck stops here." If there's something going wrong within the team, it's your job to straighten it out, eliminate the bottleneck, or remove the obstacle. It's on you, it's your neck, because it's your team.

Except, as new managers often find out, the fires never end. The great quote from the classic movie, *Men in Black* comes to mind: "There's always an Arquillian Battle Cruiser, or a Corillian Death Ray, or an intergalactic plague that is about to wipe out all life on this miserable little planet, and the only way these people can get on with their happy lives is that they DO NOT KNOW ABOUT IT!" Like the galactic agency knows, the truth of management is there's always a fire waiting.

Management 101

This brings us to the real truth of management, and although it's not a deep or hard truth to grasp, it's a truth that many will spend their entire careers seeking to master, and which will cause many more to choose never to manage. The simple, honest truth of management is that unlike your previous positions as an individual contributor, management is not, and never will be, about you. Management is about your team: making them better, making them smarter, making them shine in front of anybody else, and so on. Your job, in many ways, is to hire the best, give them what they need, and then get the hell out of their way.

"Sure, sure, that's what I'm doing when I'm out there fighting fires," the new manager tells me. "If I don't get those fires out of their way, they can't do their job." It's not entirely true—some fires, sure, your team won't have the ability to deal with, but not all fires will be ones that require your presence. As hard as it may be to understand, sometimes the best way to fight a fire is to stand back and do nothing, and let your team do what they are paid to do. Fighting fires is often a signal that the new manager wants to *do* something, not just stand back and watch. But if the team has the right skills to deal with the issue, then the manager's job is to do exactly that: stand back, watch the team do its thing, and look for ways to help the team do it better next time.

There's a lot more to be said about management—whole forests have been sacrificed to the topic—but it all comes back to the central core issue that management is not about you, it's about your team. Embrace that, and you'll be taking the first steps to becoming a great manager. Or, at least, recognizing a good one when you have one.

Ted Neward

CODE COMPILERS

CODE

Mar/Apr 2018
Volume 19 Issue 2

Group Publisher
Markus Egger

Associate Publisher
Rick Strahl

Editor-in-Chief
Rod Paddock

Managing Editor
Ellen Whitney

Content Editor
Melanie Spiller

Writers In This Issue
Kevin S. Goff
Sahil Malik
Ted Neward
Paul D. Sheriff
Mike Yeager
Chris Kinsman
Wei-Meng Lee
John V. Petersen
Rick Strahl

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.
International Bonded Couriers (IBC)
Newsstand: Ingram Periodicals, Inc.
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 $44.99. 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 300, Spring, Texas 77379
Phone: 832-717-4445
Fax: 832-717-4460

On Managers

For an industry that prides itself on its analytical ability and abstract mental processing, we often don't do a great job applying that mental skill to the most important element of the programmer's tool chest—that is, ourselves. What, exactly, does your boss do for you? For many years, stretching all the way back to the beginning of time (or so it seems), software developers have writhed under the lash of the software development manager. Managers, it seems, are responsible for all things "wrong" in a software developer's life: the impossible deadlines, the highly-misplaced budgets (plenty of money for buying an enterprise license for a tool that does nothing and that nobody will actually use, but no way, no budget for that training course), the promises made to the Sales folks that couldn't ever be built, and so on. Everywhere we turn, the ills of software development always seem to stem from the universally loathed "boss" who seemingly controls all aspects of the developer's life: what they're working on, when they're working on it, what they get paid to work on it, and even, sometimes, how they do the work. Is there no relief from this madness, this curse, foisted upon us by the clueless gods of corporation?

One company, Zappos, even went so far as to take the radical step of eliminating all corporate hierarchy. This action gathered a lot of attention and press. Roger Hodge, writing in The New Republic, called it "a radical experiment…to end the office workplace as we know it." Known as "holacracy", the intent here is simply to cut out the middle management, and let employees make strategic decisions as they see fit, capturing all of those decisions in a centralized application for visibility. What software developer wouldn't love this idea?

As it turned out, lots of them didn't: Zappos' employee turnover rate rose 10 points over the calendar year in which they executed on the holacratic model, as reported by The Atlantic in January of 2016:

…[holacracy] has led to what's being called a Zappos exodus, as 18 percent of the company's staff have taken buyouts in the last 10 months. That takes Zappos' turnover rate for 2015 to 30 percent, which is 10 percentage points above their typical annual attrition rate.

Granted, this is a solitary data point, but elsewhere (Medium, for example, was also a huge proponent of it, and ran into equally disappointing results), holacracy has yielded the same results—tremendous hope and excitement, only to drop the concept within a few years.

What is it, exactly, that management does? Why would its absence suddenly start causing employees to leave or, worse, projects to fail and companies to suffer huge setbacks?

Management Myths

When a rookie manager begins their first management job, they often bring with them a set of myths about management that they've carried around with them while not being a manager—in fact, these myths are so pervasive among the new-manager community, I'd submit that they're myths that most employees have about their management in general. Within these myths—and the debunking that follows—lie the seeds of what it is that a manager actually does, so let's walk through a few of them.

The Myth of Authority

Often, because "the boss" gets to tell their team what to work on, there is a presumption of authority that we assume comes with the title. The thinking goes, "Once I become a manager, I will be able to exercise that authority, and make the kinds of decisions that need to be made, instead of having to do all these stupid things that my manager currently has me doing." Underlying that myth is a sense of decision-making, that now it's the manager's ability to "call the shots" and tell everybody what to do. And, in particular, when that manager isn't a developer, having them make decisions about which frameworks to use or platforms to adopt just doesn't make sense to anybody. "Once I become the manager," so the thinking goes, "no longer will I be burdened by the unreasonable demands of others."

Alas, reality is far, far more complicated than that. It would be easy to simply point out that most managers in turn report to their own manager, and so decisions simply "roll downhill," as the saying goes. Thus, for real authority, one must be at the top of the management chain. But even then, authority isn't all found at the top—the CEO reports to the Board, and the Board reports to the shareholders. More eloquently, however, it's important to realize that ultimately everybody at the company reports to the customers, because if the customers stop buying the product, the company isn't long for this world regardless of who holds what title.

What's worse, new managers often find that interdependencies against other departments and/or organizations (inside or outside the company) often hem them in far more than anything their management chain could impose. That ridiculous and impossible deadline? Turns out that was imposed by Marketing because if they want to run TV spots, they need to have screen shots a full six months before the spot airs. Why? Who knows—that's what the local TV station told them. The budget for training? Slashed because the company is tight on funds. But the enterprise tool the company could afford? That turned out to be a freebie offered by the vendor as a loss-leader to try to gain a foothold inside the company. Or it was a part of a site license deal done by the parent company so it's free to us. Or so on.

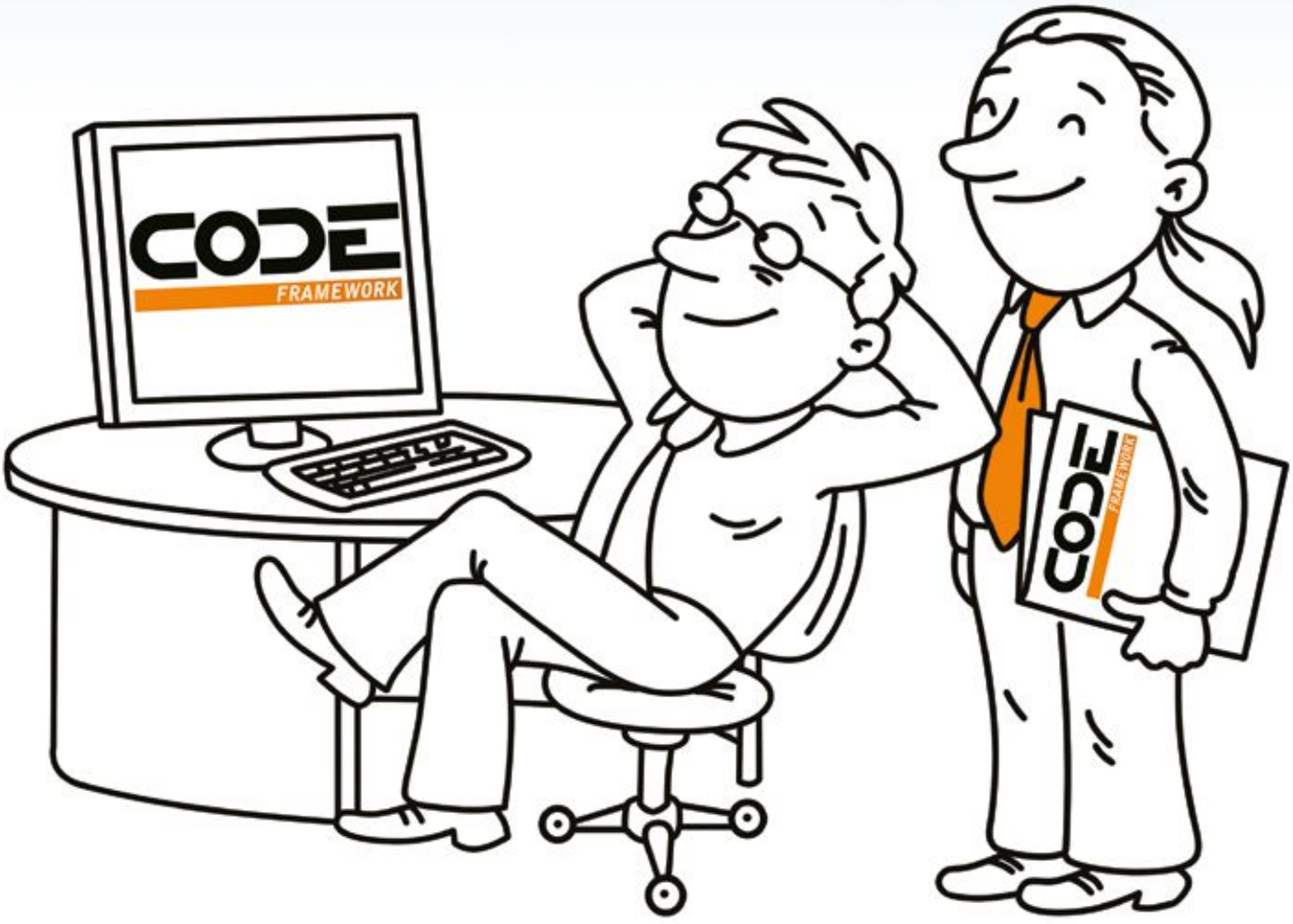
More deeply lies the hidden danger of authority—in addition to having the power to make decisions, so long as it fits within the restrictions imposed by all the interdependencies, you also have the responsibility to own the consequences of that decision. You have the authority to bring that industry "thought leader" in as a consultant on the project, like you always wanted—but now you also have the responsibility to justify that leader's six-digit consulting bill, and the corresponding realization that your budget can't afford bonuses for the team if you do. You can hire your close friend, because she's just amazing at doing this Web stuff—but now you also have to be responsible for firing her if she's not working out. And so on. As the manager, you, more than anything else in the company, are responsible for your employees and their continued ability to draw a paycheck.

The Myth of Control

Frequently, the new manager assumes that they're meant to issue marching orders, and that their team is expected to comply with said orders. That might work in the armed forces, but it definitely doesn't hold much water with the group of highly creative individuals that software development teams are encouraged to be. Think about it—when you had a boss who told you what to do, how often did you leap to it, eagerly and enthusiastically? Certainly, there are those infrequent times when the boss instructs you to do what you wanted to do anyway, but the rest? Bah. You'll do it because the boss told you to, maybe, but as soon as something

(Continued on page 73)

CODE Framework:
Business Applications Made Easy



Architected by Markus Egger and the experts at CODE Magazine, CODE Framework is the world's most productive, maintainable, and reusable business application development framework for today's developers. CODE Framework supports existing application interoperability or can be used as a foundation for ground-up development. Best of all, it's free, open source, and professionally supported.

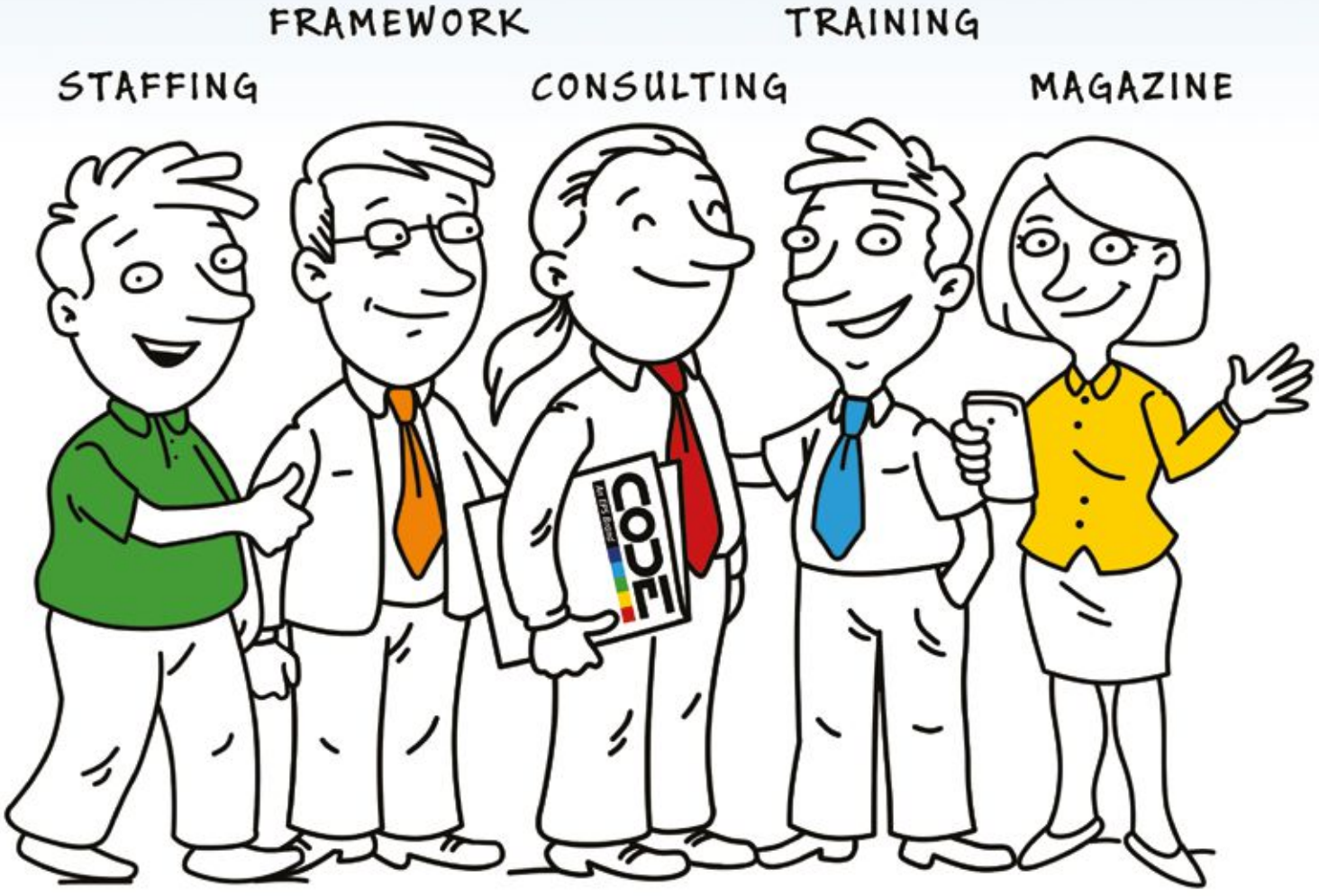
Download CODE Framework at www.codemag.com/framework

Helping Companies Build Better Software Since 1993

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

CODE
FRAMEWORK

CODE - More Than Just CODE Magazine



The CODE brand is widely-recognized for our ability to use modern technologies to help companies build better software. CODE is comprised of five divisions - CODE Consulting, CODE Staffing, CODE Framework, CODE Training, and CODE Magazine. With expert developers, a repeatable process, and a solid infrastructure, we will exceed your expectations. But don't just take our word for it - ask around the community and check our references. We know you'll be impressed.

Contact us for your free 1-hour consultation.

Helping Companies Build Better Software Since 1993

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

CODE
An EPS Brand