Angular, Machine Learning, Azure Functions, Xamarin, Python

CODE

NOV DEC 2017

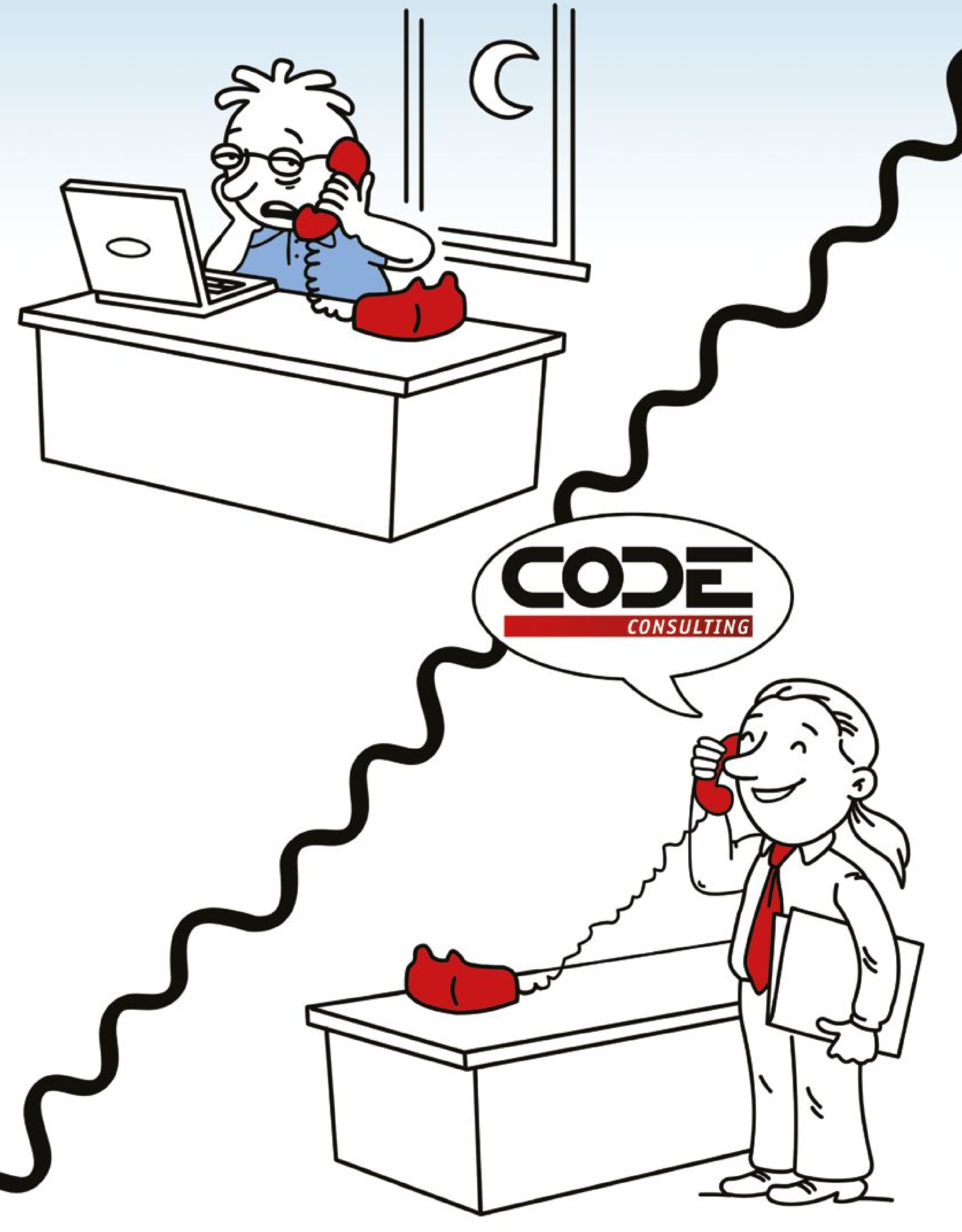
codemag.com - THE LEADING INDEPENDENT DEVELOPER MAGAZINE - US $ 5.95 Can $ 8.95

Connect to the Future with Azure Functions

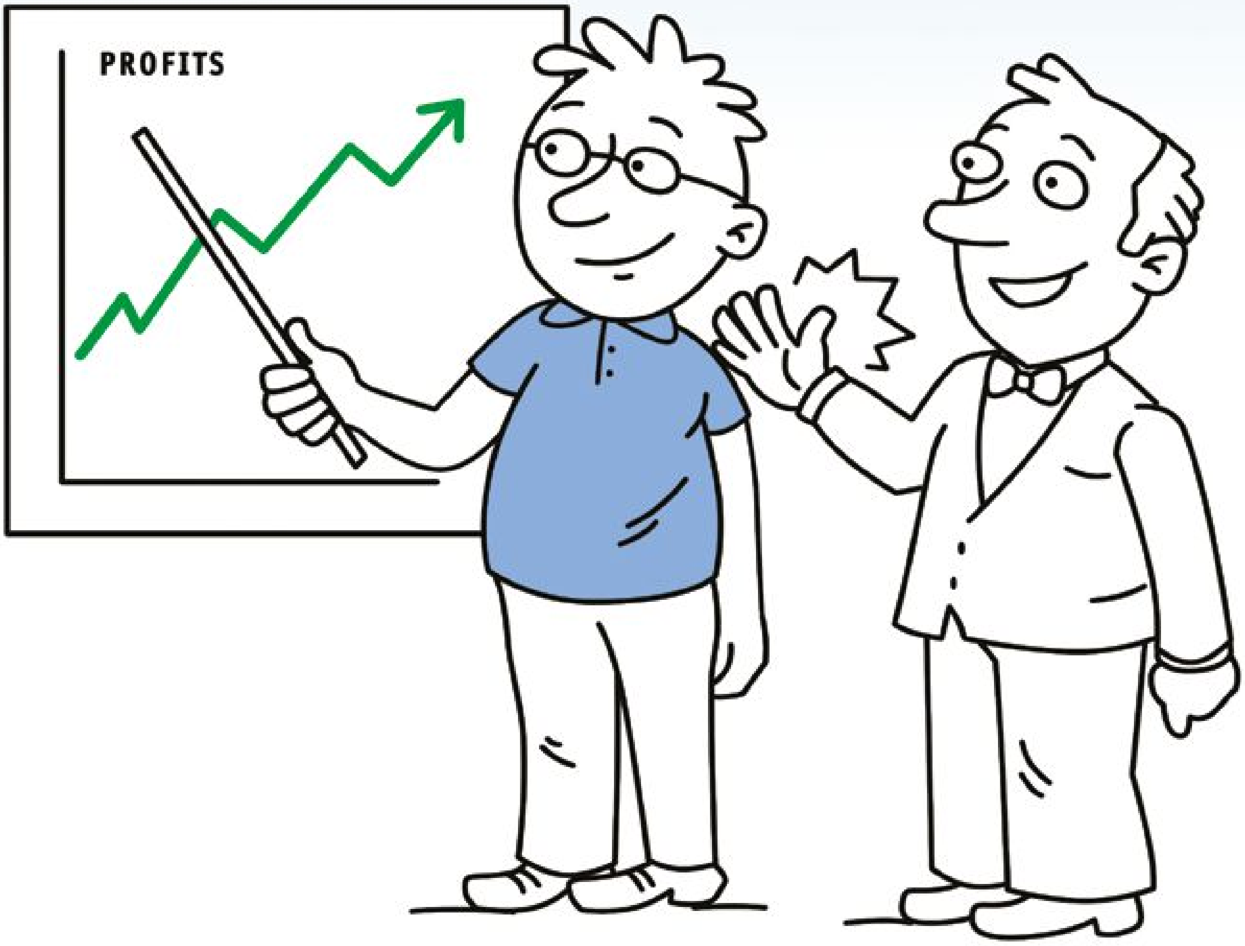
Spelunking through Legacy Code

Introduction to NativeScript

Using Python's Scikit-learn


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

Features

8 Logging in Angular Applications
Continuing his series on Angular, Paul looks at logging as a reporting tool for debugging, warnings, errors, and other types of messages, as well as calling classes to log to the console, local storage, and Web API.
Paul D. Sheriff

18 Microsoft Teams: The Developer Story
If you were wondering how to make sure that your team communicates well, you'll want to follow along closely as Sahil makes a bot using Microsoft Teams.
Sahil Malik

22 Legal Notes: Should Software Developers Be Subject to Professional Standards of Ethical Conduct?
John takes a look at what makes a person a "professional" and how the development climate is changing.
John V. Petersen

24 An Introduction to Native Android and iOS Development with NativeScript
These days, your code really must work across all of the various platforms. Unless you plan to learn a lot of languages, you'll want to apply Nic's tips for using NativeScript.
Nic Raboy

30 SQL Server Reporting Services: Eight Power Tips
Kevin looks at his favorite reporting tool (SSRS) and shows you how to maximize your users' experience.
Kevin S. Goff

38 Digging into Azure Functions: It's Time to Take Them Seriously
If you need to run discrete and small units of code in a flexible, scalable, and cost-effective manner, you need Azure functions. Jeffrey and Justin show you how to take advantage of them.
Jeffrey Palermo and Justin Self

44 Developing Cross-Platform Native Apps with a Functional Scripting Language
Vassili extends his own custom scripting language for mobile development and makes creating and placing widgets fast and easy.
Vassili Kaplan

58 Implementing Machine Learning Using Python and Scikit-learn
Instead of implementing machine learning algorithms manually, Wei-Meng found that someone else had already done the hard part. Come along as he explores a Python tool, called Scikit-learn, and builds a couple of models.
Wei-Meng Lee

66 Software Archaeology
If you've ever had to step through legacy code—even your own—you understand what a mystery some of it can be. Chris provides some tips and tricks for figuring it out.
Chris G. Williams

70 Does Anybody Really Know What Time It Is: Dates and Times across Time Zones
Since cloud computing came to town, you can't count on your system's software and appliances all being in the same time zone. Mike addresses some thorny issues and helps you keep your data up to date.
Mike Yeager

Columns

74 Managed Coder: On Professionalism
Ted Neward

Departments

6 Editorial

19 Advertisers Index

73 Code Compilers

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®
THE WORLD LEADER IN IMAGING SDKs

WORLD-CLASS MOBILE RECOGNITION SDK

Quickly and accurately extract data from any document or image with LEADTOOLS. Developers using the SDK easily add advanced OCR, Barcode, Forms, Driver's License, Passport, Check, and Credit Card recognition functionality into their applications.

Download the SDK today and start developing with the best in:

✓ RELIABILITY

✓ ACCURACY

✓ SPEED

Fully-functional sample apps built with LEADTOOLS are also available for download from **Google Play**, **App Store**, and **Microsoft Store**.

DOWNLOAD OUR 60 DAY EVALUATION
WWW.LEADTOOLS.COM

LEAD TECHNOLOGIES

SALES@LEADTOOLS.COM
800.637.1840

Lessons Learned from a Second Trip Around the Block

The last 18 months have been very interesting at Dash Point Software, Inc (my company). Our team has been involved in rewriting two major applications for a client. This has been an eye-opening experience for our team. The first of these systems is a payment-processing system that we converted from Ruby on Rails to WPF/C#. We were the original developers on this system so it made perfect sense for us to handle the conversion. The second project was to convert a business-critical website from ASP.NET Web Forms (pre-master pages, to give you a time reference) to ASP.NET/MVC. Each of these conversions had number of new features added while they were being converted.

Lesson 1: Conversions are Harder Than Meets the Eye

Converting old code can be more difficult than writing code from scratch. During this process, we grossly underestimated how much time it would take to convert code. This was particularly true for the website application. You'd think with an existing code base, it would be simple to take one set of code and convert it to another. That's a nice thought. In the span of 10+ years, lots and lots of features are added, changed, orphaned, and simply tweaked. Every one of those changes required unique and individual decisions on the part of the designers and developers. Some of the features were better documented than others. Converting this code took painstaking effort to understand the intent and to make sure that intent was managed in our new code properly. This effort took much more work than we anticipated. Which leads us to our second lesson....

Lesson 2: Set Realistic Goals

There's a trait that most developers have: We are eternal optimists. Given enough time up front, we feel like super heroes and can accomplish anything. For each project, we started with a deadline in mind and felt positive that we could accomplish the tasks at hand. In each case, we were wrong. Each project missed the deadline by six or more months. How do we, as developers, combat the chronic lateness of our projects? One idea that we're looking into is to start working on a project for a month or so before attempting to estimate the real time to completion. In one case, after a challenging four months of hardcore development, we knew we weren't going to make our deadline. At this point, we took a pause to come up with a realistic set of tasks remaining and worked on a realistic timeframe for completion. We did a much better job of estimating what was left and hit our goal within a few weeks. Setting realistic goals is a challenge for all projects and one we strive to improve.

Lesson 3: Work/Life Balance is Important

Many nights of sleep were lost on the Web project due to the effort of trying to meet the unrealistic deadline. When the team stopped to re-assess our deadline, our senior project manager told us to make sure to add in our vacation/downtime into the timeline. "Don't kill yourselves over this," I think she said. When you're knee deep in getting stuff done, the idea of downtime can slip out the window. Taking proper care of our team helped to insure that the deadlines were met while keeping our sanity.

Lesson 4: A Calm Team Gets It Done!

The day had finally arrived: We were going to ship our product! We turned off the old code at 9:00pm and started on the task of turning the new code. The team expected to be done by 1 or 2:00am, max. Well, at 11:30am the following day, we were still there. Every deployment has problems. Configurations were missed, network settings were incorrect, data conversions took longer than anticipated, files get missed, etc. As the saying goes, stuff happens! It's the attitude of the team that makes all the difference. There are two courses of action. We can freak out and declare that the sky is falling or we can remain calm and work through the issues. Our team did a marvelous job of the latter. We remained calm and as each situation presented itself, we took careful measure of the issue and set about fixing the problems. One by one, we fixed our issues and the software stabilized. Not all of the issues were resolved in the first 48 hours. Some of them took days—and in some cases weeks—to fix properly. Eventually, we shored our systems up and things are progressing well today. I firmly believe that the team's calm and measured demeanor helped us greatly.

Lesson 5: Monitoring Tools Are Invaluable

One of the saving graces in this deployment was the tools we used to monitor our software. We used a lot of them. Our SQL Server gurus used monitoring tools to tweak queries that were fast in UAT (User Acceptance Testing) but slow in production; we used other custom scripts to watch our data connections and other metrics. Finally, we used a tool that monitored the error rate of our application. This tool helped target unseen errors in our application, helping us further improve our systems. Without these tools, it would have been much more difficult to stabilize our systems in a realistic timeframe.

The Final Lesson

When building and deploying software, it's important at the end of the project to take a bit of time to reflect on what went right and what went wrong. Spend the time being introspective. Taking the time to do post-mortems on projects will help you improve your process and insure that the next one goes better.

[illegible]

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

Logging in Angular Applications

Programmers frequently use console.log to record errors or other informational messages in their Angular applications. Although this is fine while debugging your application, it's not a best practice for production applications. As Angular is all about services, it's a better idea to create a logging service that you can call from other services and components.

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.

In this logging service, you can still call console.log, but you can also modify the service later to record messages to store them in local storage or a database table via the Web API.

In this article, you'll build up the logging service in a series of steps. First, you create a simple log service class to log messages using console.log(). Next, you add some logging levels so you can report on debug, warning, error, and other types of log messages. Then you create a generic logging service class to call other classes to log to the console, local storage, and a Web API. Finally, you create a log publishing service that reads a JSON file to choose which log service classes to use.

A Simple Logging Service

To get started, create a very simple logging service that only logs to the console. The point here is to replace all of your console.log statements in your Angular application with calls to a service. Bring up an existing Angular application or create a new one. If you don't already have one, add a folder named **shared** under the \src\app folder. Create a new TypeScript file named **log.service.ts**. Add the code shown in the following snippet.

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

@Injectable()
export class LogService {
  log(msg: any) {
    console.log(new Date() + ": "
      + JSON.stringify(msg));
  }
}
```

This code creates an injectable service that can be created by Angular and injected into any of your Angular classes.

Listing 1: The LogTestComponent to test the LogService class

```
import { Component } from "@angular/core";
import { LogService }
  from '../shared/log.service';

@Component({
  selector: "log-test",
  templateUrl: "./log-test.component.html"
})
export class LogTestComponent {
  constructor(private logger: LogService) {
  }

  testLog(): void {
    this.logger.log("Test the log() Method");
  }
}
```

The log() method accepts a message that can be any type. A new date is created so each message can be logged to the console with the date and time attached to it. The date/time is not that important when just logging to the console, but once you start logging to local storage or to a database, you want the date/time attached so you know when the log messages was created. Notice the use of JSON.stringify around the msg parameter. This allows you to pass an object and it can be logged as a string.

For the purpose of following along with this article, create a new folder called \log-test and add a log-test.component.html page. Add a button to test the logging service.

```
<button (click)="testLog()">
  Log Test
</button>
```

Create a log-test.component.ts TypeScript file and add the code shown in **Listing 1** to respond to the button click event.

Add a logger variable to your constructor so Angular can inject this service into this component. Notice that in the testLog() method you now call the this.logger.log() instead of console.log(). The result is the same (see **Figure 1**) in that the message appears in the console window. However, you've now given yourself the flexibility to log this message to local storage, to a database table, to the console, or to all three. And, the best part is, you don't have to change any code in your application, other than the code in the LogService class. To use this service in your Angular application, you need to import it in your app.module.ts file. Also import the LogTestComponent you created as well.

```
import { LogService }
  from './shared/log.service';
import { LogTestComponent }
  from './log-test/log-test.component';
```

Add the LogService to the providers property in the @NgModule statement. Add the LogTestComponent class to the declarations property, as shown in the code snippet below.

```
@NgModule({
  imports: [BrowserModule],
  declarations: [AppComponent,
                 LogTestComponent],
  bootstrap: [AppComponent],
  providers: [LogService]
})
export class AppModule { }
```

Add the <log-test></log-test> selector on one of your pages to display the Log Test button. Run the application and click on the Log Test button and you should see the message appear in the console window in the F12 tools of your browser, as shown in **Figure 1**.

Different Types of Logging

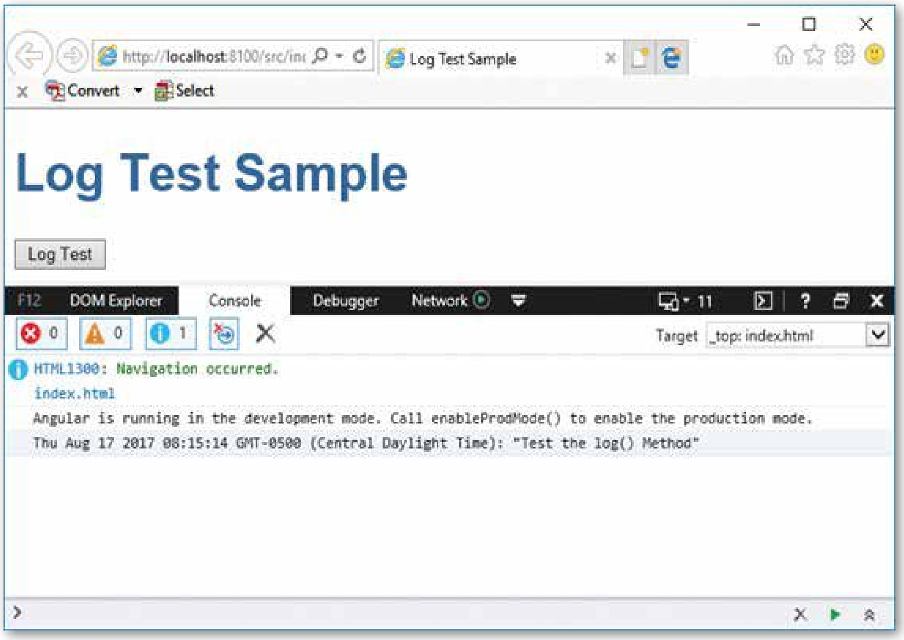
There are times when you might want only certain types of logging turned on when running your application. Many logging systems in other languages allow you to log debug messages, informational messages, warning messages, etc. Add this same ability to your LogService class by adding an enumeration and a property that you can set to control which messages to display. First, add a LogLevel enumeration in the log.service.ts file to keep track of what kind of logging to perform. Add this enumeration just after the import statement within the log.service.ts the file. Don't add it within the LogService class.

```
export enum LogLevel {
  All = 0,
  Debug = 1,
  Info = 2,
  Warn = 3,
  Error = 4,
  Fatal = 5,
  Off = 6
}
```

Add a property to the LogService class named **level** that's of the type LogLevel. Default the value to the All enumeration. While you are adding properties, add a Boolean property named **logWithDate** to specify whether you wish to add the date/time to the front of your messages or not.

```
level: LogLevel = LogLevel.All;
logWithDate: boolean = true;
```

Instead of having to set the **level** property prior to calling your logger.log() method, add the new methods debug, info, warn, error, and fatal to the LogService class (**Listing 2**). Each one of these methods calls a writeToLog() method passing in the message, the appropriate enumeration value and an optional parameter array. Delete the log() method you wrote previously and replace that one method with all the methods from **Listing 2**.



```
HTML1300: Navigation occurred.
index.html
Angular is running in the development mode. Call enableProdMode() to enable the production mode.
Thu Aug 17 2017 08:15:14 GMT-0500 (Central Daylight Time): "Test the log() Method"
```

Figure 1: Sample of using the LogService

The optional parameter array means you can pass any parameters you want to be logged. For example, any of the following calls are valid.

```
this.logger.log("Test 2 Parameters",
                "Paul", "Smith");

this.logger.debug("Test Mixed Parameters",
                  true, false, "Paul", "Smith");

let values = ["1", "Paul", "Smith"];
this.logger.warn("Test String and Array",
                 "Some log entry", values);
```

The writeLog() method, **Listing 3**, checks the level passed by one of the methods against the value set in the **level** property. This **level** property is checked in the shouldLog() method. Both of these methods should now be added to your LogService class.

The shouldLog() method determines if logging should occur based on the **level** property set in the LogService class. This

Listing 2: Add methods to your LogService class to write different kinds of messages

```
debug(msg: string, ...optionalParams: any[]) {
  this.writeToLog(msg, LogLevel.Debug,
                  optionalParams);
}

info(msg: string, ...optionalParams: any[]) {
  this.writeToLog(msg, LogLevel.Info,
                  optionalParams);
}

warn(msg: string, ...optionalParams: any[]) {
  this.writeToLog(msg, LogLevel.Warn,
                  optionalParams);
}

error(msg: string, ...optionalParams: any[]) {
  this.writeToLog(msg, LogLevel.Error,
                  optionalParams);
}

fatal(msg: string, ...optionalParams: any[]) {
  this.writeToLog(msg, LogLevel.Fatal,
                  optionalParams);
}

log(msg: string, ...optionalParams: any[]) {
  this.writeToLog(msg, LogLevel.All,
                  optionalParams);
}
```

service is created as a singleton by Angular, so once this **level** property is set, it remains that value until you change it in your application. The shouldLog() checks the parameter passed in against the **level** property set in the LogService class. If the level passed in is greater than or equal to the **level** property, and logging is not turned off, then a true value is returned from this method. A true return value tells the writeToLog() method to log the message.

```
private shouldLog(level: LogLevel): boolean {
  let ret: boolean = false;

  if ((level >= this.level &&
       level !== LogLevel.Off) ||
       this.level === LogLevel.All) {
    ret = true;
  }

  return ret;
}
```

There's one more method call in the writeToLog() method called formatParams(). This method is used to create a comma-delimited list of the parameter array. If all parameters in the array are simple data types and not an object, then the local variable named **ret** is returned after the join() method is used to create a comma-delimited list from the array. If there is one object, loop through each of the items in the params array and build the **ret** variable using the JSON.stringify() method to convert each parameter to a string, and then append a comma after each.

```
private formatParams(params: any[]): string {
  let ret: string = params.join(",");

  // Is there at least one object in the array?
  if (params.some(p => typeof p == "object")) {
    ret = "";
    // Build comma-delimited string
    for (let item of params) {
      ret += JSON.stringify(item) + ",";
    }
  }

  return ret;
}
```

Create Log Entry Class

Instead of building a string of the log information, and formatting the parameters in the writeToLog() method, create a class named LogEntry to do all this for you. Place this new class within the log.service.ts file. The LogEntry class, shown in **Listing 4**, has properties for the date of the log entry, the message to log, the log level, an array of extra info to log, and a Boolean you set to specify to include the date with the log message.

Listing 3: The writeToLog() method creates the message to write into your log

```
private writeToLog(msg: string,
                   level: LogLevel,
                   params: any[]) {
  if (this.shouldLog(level)) {
    let value: string = "";

    // Build log string
    if (this.logWithDate) {
      value = new Date() + " - ";
    }
    value += "Type: " + LogLevel[this.level];
    value += " - Message: " + msg;
    if (params.length) {
      value += " - Extra Info: "
        + this.formatParams(params);
    }

    // Log the value
    console.log(value);
  }
}
```

Listing 4: Create a LogEntry class to make creating log messages easier

```
export class LogEntry {
  // Public Properties
  entryDate: Date = new Date();
  message: string = "";
  level: LogLevel = LogLevel.Debug;
  extraInfo: any[] = [];
  logWithDate: boolean = true;

  buildLogString(): string {
    let ret: string = "";

    if (this.logWithDate) {
      ret = new Date() + " - ";
    }
    ret += "Type: " + LogLevel[this.level];
    ret += " - Message: " + this.message;
    if (this.extraInfo.length) {
      ret += " - Extra Info: "
        + this.formatParams(this.extraInfo);
    }

    return ret;
  }

  private formatParams(params: any[]): string {
    let ret: string = params.join(",");

    // Is there at least one object in the array?
    if (params.some(p => typeof p == "object")) {
      ret = "";
      // Build comma-delimited string
      for (let item of params) {
        ret += JSON.stringify(item) + ",";
      }
    }

    return ret;
  }
}
```

The buildLogString() method is similar to what you wrote in the writeToLog() method earlier. This method gathers the values from the properties of this class and returns them in one long string that can be used to output to the console window. Remove the formatParams() method from the LogService class after you've built the LogEntry class. You're going to use this LogEntry class from each of the different logging classes you build in the rest of this article.

Now that you have this LogEntry class built, and have removed the formatParams() from the LogService class, rewrite the writeToLog() method to look like the code below.

```
private writeToLog(msg: string,
                   level: LogLevel,
                   params: any[]) {
  if (this.shouldLog(level)) {
    let entry: LogEntry = new LogEntry();

    entry.message = msg;
    entry.level = level;
    entry.extraInfo = params;
    entry.logWithDate = this.logWithDate;

    console.log(entry.buildLogString());
  }
}
```

After modifying the writeToLog() method, rerun the application and you should still see your log messages being displayed in the console window.

Log Publishing System

When logging exceptions, or any kind of message, it's a good idea to write those log entries to different locations in case one of those locations isn't accessible. In this way, you stand a better chance of not losing any messages. To do this, you need to create three different classes for logging. The first publisher is a LogConsole class that logs to the console window. The second publisher is Log LocalStorage to log messages to Web local storage. The third publisher is LogWebApi for calling a Web API to log messages to a backend table in a database.

Instead of hard-coding each of these classes in the LogService class, you're going to create a **publishers** property (**Figure 2**)**,** which is an array of an abstract class called LogPublisher. Each of the logging classes you create extends this abstract class.

The LogPublisher class contains one property named location. This property is used to set the key for local storage and the URL for the Web API. This class also needs two methods: log() and clear(). The log() method is overridden in each class that extends LogPublisher and is responsible for performing the logging. The clear() method removes all log entries from the data store.

Add a new TypeScript file named log-publishers.ts to the \ shared folder in your project. You're going to need a few import statements at the top of this file. In fact, you're going to add more later, but for now, just add Observable, the Observable of, and LogEntry classes. Write the code shown in the next snippet to create your abstract LogPublisher class.

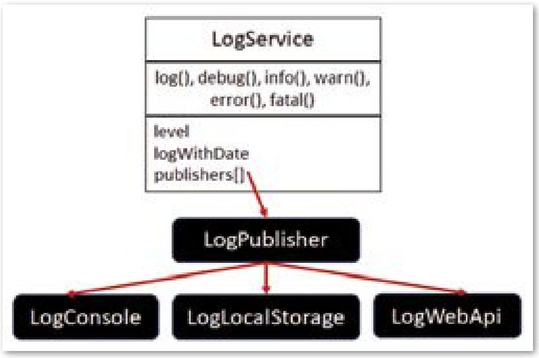


Figure 2: Create an abstract class from which all your log publishers inherit.

```
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/observable/of';

import { LogEntry } from './log.service';

export abstract class LogPublisher {
  location: string;

  abstract log(record: LogEntry):
             Observable<boolean>
  abstract clear(): Observable<boolean>;
}
```

Now that you have the template for creating each log publishing class, let's start building them.

Log to Console

The first class you create to extend the LogPublisher class writes messages to the console. You're eventually going to remove the call to console.log() from the LogService class you created earlier. LogConsole is a very simple class that displays log data to the console window using console.log(). Add the following code below the LogPublisher class in the log-publisher.ts file.

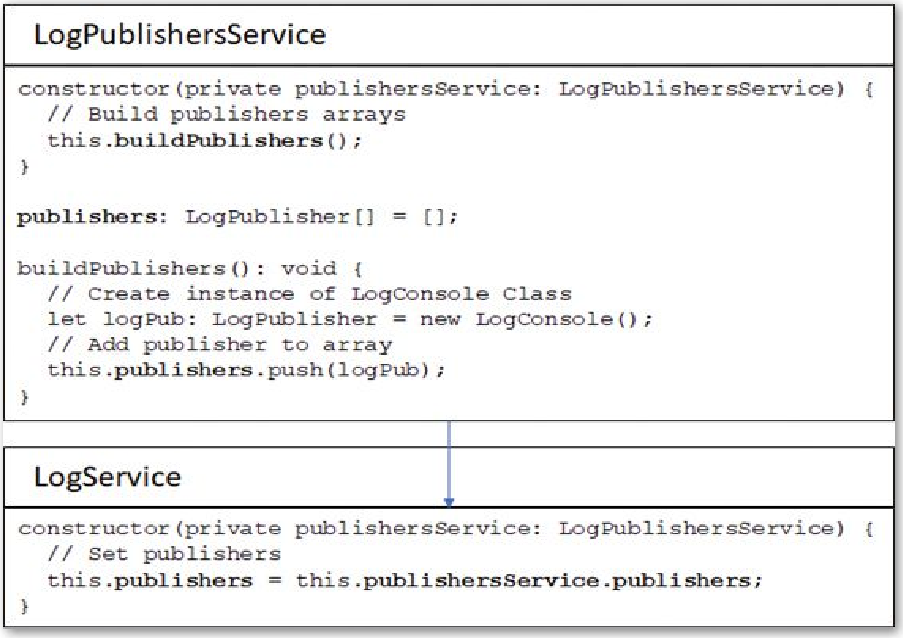
```
export class LogConsole extends LogPublisher {
  log(entry: LogEntry): Observable<boolean> {
    // Log to console
    console.log(entry.buildLogString());

    return Observable.of(true);
  }

  clear(): Observable<boolean> {
    console.clear();

    return Observable.of(true);
  }
}
```

Notice that the log() method in this class accepts an instance of the LogEntry class. This parameter coming in, named **entry**, calls the buildLogString() method to create a string of the complete log entry data to be displayed to the console window. Each log() method needs to return an observable of the type Boolean back to the caller. Because nothing can go wrong with logging to the console, just hard-code a True return value.



```
constructor(private publishersService: LogPublishersService) {
  // Build publishers arrays
  this.buildPublishers();
}

publishers: LogPublisher[] = [];

buildPublishers(): void {
  // Create instance of LogConsole Class
  let logPub: LogPublisher = new LogConsole();
  // Add publisher to array
  this.publishers.push(logPub);
}
```

```
constructor(private publishersService: LogPublishersService) {
  // Set publishers
  this.publishers = this.publishersService.publishers;
}
```

Figure 3: The LogPublishersService class builds the list of publishers that's consumed by the LogService class.

Listing 5: The LogPublishersService is responsible for creating a list of publishing objects

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

import { LogPublisher, LogConsole }
  from "./log-publishers";

@Injectable()
export class LogPublishersService {
  constructor() {
    // Build publishers arrays
    this.buildPublishers();
  }

  // Public properties
  publishers: LogPublisher[] = [];

  // Build publishers array
  buildPublishers(): void {
    // Create instance of LogConsole Class
    this.publishers.push(new LogConsole());
  }
}
```

The clear() method must also be overridden in any class that extends the LogPublisher class. For the console window, call the clear() method to clear all messages published to the console window.

The Log Publishers Service

As you saw in **Figure 2**, there's a **publishers** array property in the LogService class. You need to populate this array with instances of LogPublisher classes. The only class you've built so far is LogConsole, but soon you'll build LogLocalStorage and LogWebApi classes too. Instead of building the list of publishers in the LogService class, create another service class to build the list of log publishers. This service class, named LogPublishersService, is responsible for building the array of log publishing classes. This service is passed into the LogService class so the **publishers** array can be assigned from the LogPublishersService (**Figure 3**). At first, you're going to just hard-code each of the log classes, but later in this article, you're going to read the list publishers from a JSON file.

The LogPublishersService class (**Listing 5**) needs to be defined as an injectable service so Angular can inject it into the LogService class. In the constructor of this class, you call a method named buildPublishers(). This method creates each instance of a LogPublisher and adds each instance to the **publishers** array. For now, just add the code to create new instance of the LogConsole class and push it onto the **publishers** array.

Update AppModule Class

Like any Angular service, once you create it, you must register it in the app.module.ts file by importing it and adding it to the providers property of the @NgModule. Open app.module.ts and add the import near the top of the file.

```
import { LogPublishersService }
   from "./shared/log-publishers.service";
```

Next, add the service to the providers property in the @NgModule decorator function.

```
@NgModule({
  imports: [BrowserModule, FormsModule,
            HttpModule],
  declarations: [AppComponent,
                 LogTestComponent],
  bootstrap: [AppComponent],
  providers: [LogService, LogPublishersService]
})
```

Modify the LogService Class

It's now time to modify the LogService class to use this LogPublishersService class. Open the log.service.ts TypeScript file and add two import statements near the top of the file.

```
import { LogPublisher } from "./log-publishers";
import { LogPublishersService }
   from "./shared/log-publishers.service";
```

Add a property named **publishers** that is an array of LogPublisher types.

```
publishers: LogPublisher[];
```

Add a constructor to the LogService class so Angular injects the LogPublishersService. Within this constructor, take the **publishers** property from the LogPublishersService and assign the contents to the **publishers** property in the LogService class.

```
constructor(private publishersService:
                    LogPublishersService) {
  // Set publishers
  this.publishers =
    this.publishersService.publishers;
}
```

Locate the writeToLog() method and remove the following line of code from this method.

```
console.log(entry.buildLogString());
```

Where you removed the above line of code, add a **for loop** to iterate over the list of publishers. Each time through the loop, invoke the log() method of the logger, passing in the LogEntry object. Because the log() method returns an observable, you should subscribe to the result and write the Boolean return value to the console window.

```
for (let logger of this.publishers) {
  logger.log(entry)
    .subscribe(response =>
             console.log(response));
}
```

Once again, run the application and click the Test Log button to see a log entry written to the console window. You have added a few classes and a service only to publish to the console window; you should be able to see the advantages of this kind of approach. You can now add new LogPublisher classes, add them to the array in the LogPublishersService class, and you're now publishing to an additional location.

Log to Local Storage

The next publisher to add is one that stores an array of LogEntry objects into your Web browser's local storage. Open the log-publishers.ts file and add the code shown in **Listing 6** to this file. The LogLocalStorage class needs to set a key value for setting the items into local storage. Use the **location** property to set the key value to use. In this case, the **location** is set in the constructor. When you have a constructor in a derived class, you always need to call the super() method in order to invoke the constructor of the base class.

Local storage allows you to store quite a bit of data, so let's add each log entry each time the log() method is called. The setItem() method is used to set a value into local storage. If you call setItem() and pass in a value, any old value in the key location is replaced with the new value. Read the previous values first from local storage using the getItem() method. Parse that into an array of LogEntry objects, or if there was no value stored in that key location, return an empty array. Push the new LogEntry object onto the array, stringify the new array, and place the stringified array into local storage.

One note of caution on local storage; there's a limit set by each browser to how much data can be stored. The limits vary between browsers, and as of this writing, it varies from 2MB to 10MB. You might want to consider writing some additional code in the catch block of the log() method to remove the oldest values from the array prior to storing the new log entry.

The clear() method is used to clear local storage at the specified key location. Call the removeItem() method of the localStorage object to clear all values within this location.

Now that you have your new class to store log entries into local storage, you need to add this to the publishers array. Open the log-publishers.service.ts file and modify the import statement to include your new LogLocalStorage class.

```
import { LogPublisher, LogConsole,
         LogLocalStorage }
  from "./log-publishers";
```

Modify the buildPublishers() method of the LogPublishersService class to create an instance of the LogLocalStorage class and push it onto the publishers array as shown in the following code.

```
buildPublishers(): void {
  // Create instance of LogConsole Class
  this.publishers.push(new LogConsole());

  // Create instance of LogLocalStorage Class
  this.publishers.push(new LogLocalStorage());
}
```

Listing 6: Create a LogLocalStorageService class to store messages into local storage

```
export class LogLocalStorage
              extends LogPublisher {
  constructor() {
    // Must call super() from derived classes
    super();
    // Set location
    this.location = "logging";
  }

  // Append log entry to local storage
  log(entry: LogEntry): Observable<boolean> {
    let ret: boolean = false;
    let values: LogEntry[];

    try {
      // Get previous values from local storage
      values = JSON.parse(
           localStorage.getItem(this.location))
              || [];
      // Add new log entry to array
      values.push(entry);

      // Store array into local storage
      localStorage.setItem(this.location,
                          JSON.stringify(values));

      // Set return value
      ret = true;
    } catch (ex) {
      // Display error in console
      console.log(ex);
    }

    return Observable.of(ret);
  }

  // Clear all log entries from local storage
  clear(): Observable<boolean> {
    localStorage.removeItem(this.location);
    return Observable.of(true);
  }
}
```

You can now re-run the application and you should be logging to both the console window and to local storage. To test this, set a break point in the log() method of the LogLocalStorage class and see if it retrieves the previous values that you logged into local storage.

Log to Web API

The last logging class you are going to create is one to send an instance of the LogEntry class to a Web API method. From this Web API you could then write code to store the log entry into a database table. I'm not going to provide you with a table—I'll leave that to you. I'm using Visual Studio and C# to create my Web API calls, so I'm going to add a C# class to my project that's the same name and has the same properties as the LogEntry class I created in Angular.

```
public class LogEntry
{
  public DateTime EntryDate { get; set; }
  public string Message { get; set; }
  public LogLevel Level { get; set; }
  public object[] ExtraInfo { get; set; }
}
```

I'm also adding a C# enumeration to my project to map to the TypeScript LoggingLevel enumeration.

```
public enum LogLevel
{
  All = 0,
  Debug = 1,
  Info = 2,
  Warn = 3,
  Error = 4,
  Fatal = 5,
  Off = 6
}
```

Finally, I'm going to add a Web API controller class (**Listing 7**) to my project. This class has one method at this point to allow Angular to post the LogEntry record to this method. It's in this Post() method that you write the code to store the log data into a database table. For purposes of this article, I'm just going to return a result of OK (true) back to the caller.

Now that you have the Web API classes created, you can go back to the log-publishers.ts file and add some import statements for calling a Web API. Add the following import statements near the top of this file.

```
import { Http, Response,
         Headers, RequestOptions }
  from '@angular/http';
import 'rxjs/add/operator/map';
import 'rxjs/add/operator/catch';
import 'rxjs/add/observable/throw';
```

Add the LogWebApi class shown in **Listing 8** at the bottom of the log-publishers.ts file. The constructor for this class is very similar to the one you wrote for the LogLocalStorage class. You do need to include the HTTP service as you're going to need this to call the Web API. You also must call super() to execute the constructor of the base class. Finally, set the **location** property to the URL of the Web API call.

The log() method accepts a LogEntry object that's sent to the Web API method. Because you're performing a POST to the Web API, you need to create the appropriate headers to specify the content type you're sending as application/json. The post() method on the Angular HTTP service is called to pass the LogEntry object to the Web API class you created.

You also need a clear() method in this class to override the abstract method in the base class. In order to keep the length of this article shorter, I'm not showing how to do clear log entries, but it's similar to the log() method. You call a Web API method that writes the appropriate SQL to delete all rows from your log table in your database.

Open the log-publishers.service.ts file and modify the import statement to include your new LogWebApi class.

```
import { LogPublisher, LogConsole,
         LogLocalStorage, LogWebApi }
  from "./log-publishers";
```

Open your log-publishers.service.ts file and add an import for the HTTP service near the top of the file.

```
import { Http } from '@angular/http';
```

Next, add the HTTP service to the constructor of this class.

```
constructor(private http: Http) {
  // Build publishers arrays
  this.buildPublishers();
}
```

Finally, in the buildPublishers() method, create a new instance of the LogWebApi class and pass in the HTTP service as shown in the code below.

```
buildPublishers(): void {
  // Create instance of LogConsole Class
  this.publishers.push(new LogConsole());
```

Sample Code

You can download the sample code for this article on the CODE Magazine website or by visiting my website at http://www.pdsa.com/downloads. Select PDSA Articles, and then select "Code Magazine - Logging in Angular Applications" from the drop-down list.

Listing 7: The LogController allows you to store log entries via a Web API call

```
public class LogController : ApiController
{
  // POST api/<controller>
  [HttpPost]
  public IHttpActionResult Post(
              [FromBody]LogEntry value)
  {
    IHttpActionResult ret;

    // TODO: Write code to store logging
    // data in a database table

    // Return OK for now
    ret = Ok(true);

    return ret;
  }
}
```

Listing 8: Create a LogWebApiService class to call a Web API for tracking log messages

```
export class LogWebApi extends LogPublisher {
  constructor(private http: Http) {
    // Must call super() from derived classes
    super();
    // Set location
    this.location = "/api/log";
  }

  // Add log entry to back end data store
  log(entry: LogEntry): Observable<boolean> {
    let headers = new Headers(
     { 'Content-Type': 'application/json' });
    let options = new
      RequestOptions({ headers: headers });

    return this.http.post(this.location,
                           entry, options)
      .map(response => response.json())
      .catch(this.handleErrors);
  }

  // Clear all log entries from local storage
  clear(): Observable<boolean> {
    // TODO: Call Web API to clear all values
    return Observable.of(true);
  }

  private handleErrors(error: any):
                Observable<any> {
    let errors: string[] = [];
    let msg: string = "";

    msg = "Status: " + error.status;
    msg += " - Status Text: " + error.statusText;
    if (error.json()) {
      msg += " - Exception Message: " +
             error.json().exceptionMessage;
    }
    errors.push(msg);

    console.error('An error occurred', errors);

    return Observable.throw(errors);
  }
}
```

```
  // Create instance of LogLocalStorage Class
  this.publishers.push(new LogLocalStorage());

  // Create instance of LogWebApi Class
  this.publishers.push(
    new LogWebApi(this.http));
}
```

You need to register the HTTP service with your AppModule. Open app.module.ts and add the following import near the top of this file.

```
import { HttpModule } from '@angular/http';
```

Add the HttpModule to the imports property in the @NgModule() function decorator.

```
imports: [BrowserModule, HttpModule],
```

Now that you have this new publisher added to the array, you should be able to run your logging application, and when you click on the Log Test button, you should see that it's making the call to your Web API method.

Read Publishers from JSON File

Open the log-publishers.ts file and add a new class called LogPublisherConfig. This class is going to hold the individual objects read from a JSON file you see in **Listing 9**.

```
class LogPublisherConfig {
  loggerName: string;
  loggerLocation: string;
  isActive: boolean;
}
```

Build the JSON file by adding an \assets folder underneath the \src\app folder. Add a JSON file called log-publishers.json in the \assets folder and add the code from **Listing 9**. Each of the object literals in this JSON array relate to one of the classes you created for logging to the console, local storage, and the Web API.

Add a few more import statements to your log-publishers.service.ts file.

```
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/operator/map';
import 'rxjs/add/operator/catch';
import 'rxjs/add/observable/throw';
```

Add a constant just after these imports, to point to this file.

```
const PUBLISHERS_FILE =
  "/src/app/assets/log-publishers.json";
```

In the LogPublishersService class, add a new method named handleErrors (**Listing 10**) to take care of any errors that might happen during any HTTP service calls. Also, in the LogPublishersService class, create a new method to read the data from this JSON file. Let's call this method getLoggers(). Because you already injected the HTTP service into this class, you can use this service to read from the JSON file.

```
getLoggers(): Observable<LogPublisherConfig[]> {
  return this.http.get(PUBLISHERS_FILE)
    .map(response => response.json())
    .catch(this.handleErrors);
}
```

Now that you have this method to return the array from this file, modify the buildPublishers() method to subscribe to this Observable array of LogPublisherConfig object. The new code for the buildPublishers() method is shown in **Listing 11**.

The buildPublishers() method calls the getLoggers() method and subscribes to the output from this method.

Listing 9: Create a LogLocalStorageService class to store messages into local storage

```
[
  {
    "loggerName": "console",
    "loggerLocation": "",
    "isActive":  true
  },
  {
    "loggerName": "localstorage",
    "loggerLocation": "logging",
    "isActive": true
  },
  {
    "loggerName": "webapi",
    "loggerLocation": "/api/log",
    "isActive": true
  }
]
```

Listing 10: You always should have a method to handle errors when using the HTTP service

```
private handleErrors(error: any):
           Observable<any> {
  let errors: string[] = [];
  let msg: string = "";

  msg = "Status: " + error.status;
  msg += " - Status Text: " + error.statusText;
  if (error.json()) {
    msg += " - Exception Message: "
        + error.json().exceptionMessage;
  }
  errors.push(msg);

  console.error('An error occurred', errors);

  return Observable.throw(errors);
}
```

Listing 11: Create your array of publishers by reading the values from a JSON file

```
buildPublishers(): void {
  let logPub: LogPublisher;

  this.getLoggers().subscribe(response => {
    for (let pub of response.filter(p => p.isActive)) {
      switch (pub.loggerName.toLowerCase()) {
        case "console":
          logPub = new LogConsole();
          break;
        case "localstorage":
          logPub = new LogLocalStorage();
          break;
        case "webapi":
          logPub = new LogWebApi(this.http);
          break;
      }
      // Set location of logging
      logPub.location = pub.loggerLocation;
      // Add publisher to array
      this.publishers.push(logPub);
    }
  });
}
```

The output is an array of LogPublisherConfig objects. The array of configuration objects is filtered to only loop through those that have their isActive property set to a True value. For each iteration through the loop, check the loggerName property and compare that value with those listed in each case statement. If a match is found, a new instance of the corresponding LogPublisher class is created. The loggerLocation property is set to the location property of each LogPublisher class. The newly instantiated publisher object is then added to the **publishers** array property. As all of this happens in the constructor of this service class; the publishers array is already set to the list of publishers to use by the time it is injected into the LogService class. You should be able to run the Angular application and click on the Log Test button and see the log messages published to all publishers marked as isActive in the JSON file.

Summary

It's always a best practice to log messages as you move throughout your Angular applications. Exceptions should always be logged, but you may also wish to log the debug, warning, and informational messages as well. Creating a flexible logging system like the one presented in this article assists with this best practice. Using a configuration JSON file to store which publishers you wish to log to saves having to hard-code publishers within your log service class.

Paul D. Sheriff

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

Microsoft Teams: The Developer Story

Yes, I know what you're thinking! Another day, another product. As a consultant with limited time and resources, I really must pick and choose my battles. After all, what I choose to do also decides what I won't be able to do. In that vein, I think that Microsoft Teams, the newest collaboration product in Office 365, deserves a look. Microsoft Teams warrants a look for both business users and developers. I'll keep this article focused on developers.

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.

What is Microsoft Teams?

At a fundamental level, Microsoft Teams is a collaboration product. It's integrated into Office 365, but it's a new product built from the ground up. It can be used in a browser, on your phone, or as a desktop application. If you were wondering, yes, it's internally built using Web-based technologies such as Electron, Cordova or similar, and the usual Web stack, such as HTML, JavaScript, etc.

What's compelling about Teams is that it feels like a modern product. It doesn't feel like a dowdy old product rooted in 2001 that doesn't scale to modern organization needs or work properly—or not at all—on a Mac. It's ready for today's information worker! Ugh, did I just use that term again, "information worker?" Let's get back to the developer story.

The Developer Story for Teams

One way to think of teams is chatrooms. It's more than that, of course, but at a very high level, you create channels (a.k.a chatrooms) within your organization where people can communicate with each other. Users can also engage in a 1:1 conversation, upload files to share with their team, or have applications push in information in numerous forms.

And that's where you come in! You're the developer, and you can extend teams in the following ways:

- **Tabs:** Provides a dedicated canvas that lets team members access your service while connected to a channel or private chat.
- **Bots:** Built on the Microsoft Bot Framework, allows team members to interact with your service via conversations.
- **Connectors**: Allow your services to send notifications into channels. These notifications can include rich actionable messages that can expose some user interface, allowing users to interact directly with your service through a channel. I covered this in the May/June issue of CODE Magazine (http://www.codemag.com/Article/1705031).
- **Extensions:** Allow you to share your app's content directly into team conversations.
- **Sending messages:** Permits contact directly into a user's activity feed.

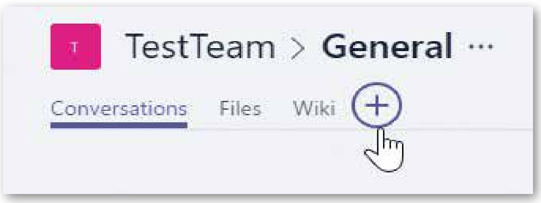


Figure 1: Adding a tab

Tabs

Tabs allow you to embed Web pages inside a team. From a user point of view, this is extremely simple to do. Just visit any team channel, and choose to click on the + button, as shown in **Figure 1**.

Clicking on the Add button brings up a dialog box, as shown in **Figure 2,** that allows you to embed any relevant content right inside a team.

Go ahead and try embedding any tab inside your channel. These are just Web pages. You can create a simple website with anything you wish in it. It could be dashboards, detail pages; really, just about anything. And you register it using a simple manifest file. When you create this website that works as a tab, you have a choice of implementing one or two pages.

- **The configuration page:** This is optional. Configurable tabs give you the opportunity to allow the user to specify some configuration information before the tab is added. You can also allow users to update a tab after they add it via this page.
- **The content page:** This page is where the functionality of your tab lives; it's the page the user sees when the user visits the tab via a team.

You can also add tabs in one of two scopes. One way is in the team scope where you add tabs to a channel. These are currently configurable tabs only. The other way is that you can add them in personal scope, where the user interacts with the tab via the app bar. Currently, only static tabs are allowed in the personal scope.

Additionally, your website that works as a tab, i.e., the content page, must follow the following rules

- It must be hosted on https
- The content must work in an iframe, and you must set the following HTTP header:

```
Content-Security-Policy:
  frame-ancestors teams.microsoft.com
  *.teams.microsoft.com *.skype.com
```

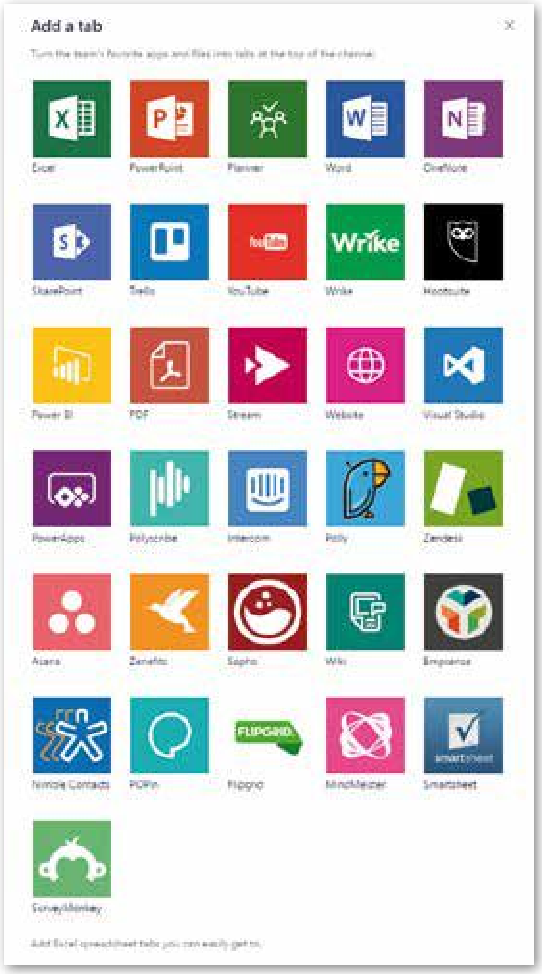
- For IE11, you also need to set the X-Content-Security-Policy header.
- Once your page loads, you need to call the Java Script method "microsoftTeams.initialize()" to show your page.
- You also need a manifest file, and all URLs being used within your tab must be under the "validDo-

mains" list in your manifest. You may reference the schema for the teams manifest file here https://msdn.microsoft.com/en-us/microsoft-teams/schemas.

As I mentioned earlier, your tab can have a configuration page specified as the "configurationUrl" parameter of the manifest file. There are some requirements for this configuration page also.

- The Save button on this page is disabled by default. The idea is that when the user has finished configuring the tab, the Save button becomes enabled. You can enable the Save button at the right moment by calling the microsoftTeams.settings.setValidityState(true) method.
- This configuration page is responsible for letting the content page know what settings it needs to run under. You can do so by calling the microsoftTeams.settings.setSettings method.
- You also have the opportunity to fire off a long-running operation when the user hits Save. You can register the event handler by calling the microsoftTeams.settings.registerOnSaveHandler method and passing in a function. The only requirement here is that you must complete this operation in 30 seconds or Microsoft Teams terminates the operation and shows an error message. The Save handler also needs to notify you of success or failure. It can do so by calling saveEvent.notifySuccess() or saveEvent.notifyFailure() methods. The saveEvent parameter is passed in as a parameter to the Save handler.

In either the content page or the configuration page, you can authenticate the user by calling the microsoftTeams.authentication.authenticate method. This may be neces-


Add a tab

Turn the team's favorite apps and files into tabs at the top of the channel.

Add Excel spreadsheet tabs you can easily get to.

Figure 2: Tabs available out of the box

Advertisers Index

CODE Consulting	www.codemag.com/techhelp	2, 67
CODE Divisions	www.codemag.com	75
CODE Framework	www.codemag.com/framework	33
CODE Magazine	www.codemag.com	57, 71
CODE Staffing	www.codemag.com/staffing	17
dtSearch	www.dtSearch.com	69
JetBrains	www.jetbrains.com	76
LEAD Technologies	www.leadtools.com	5
Melissa Global Intelligence	www.melissa.com/code	7
Tower48	www.tower48.com	49
Xamalot	www.xamalot.com	37

ADVERTISERS INDEX

Angular, Machine Learning, Azure Functions, Xamarin, Python
CODE
Connect to the Future with Azure Functions
Spelunking through Legacy Code
Introduction to NativeScript
Using Python's Scikit-learn

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

sary if the user needs to first authenticate before being able to configure a tab. For instance, you may have bug-tracking software that you wish to add as a tab. This bug-tracking software has its own authentication mechanism. The microsoftTeams.authentication.authenticate method allows you to authenticate at the bug-tracking software's URL. That URL opens in a pop-up window and is able to notify teams of success or failure. Your app can set its session cookie so the user doesn't need to sign in again.

It's perhaps also valuable to understand the context under which your tab is operating. Context contains valuable information such as team ID, channel ID, locale, theme, etc. This context can be easily received in your Web application by calling the microsoftTeams.getContext method. You can also react to theme changes by using a theme handler. This theme handler can be registered using the microsoftTeams.registerOnThemeChangeHandler method.

Bots

Next, let's talk about bots. Bots are two-way conversations, except you're talking to a computer program! Really!

It isn't as fancy as it sounds. You simply use the Microsoft Bot Framework to author a bot and register it within your team. Bots currently are supported in 1:1 chats (personal scope) or channel conversations (team scope). Group chats don't currently support bots. Bots appear like any other user except they have a hexagonal avatar icon and no mood message.

Creating a bot for a Microsoft Team takes four main steps:

1. Register the bot at https://dev.botframework.com/. If you wish to have the bot surface up in Microsoft Teams, you need to add Microsoft Teams as a channel and reuse any Microsoft App ID that you generate on the registration page. You'll need to update your app package/manifest for the bot with this App ID.
2. You need to write the bot. You can do so using .NET using the Microsoft.Bot.Connector.Teams nuget package, or the botbuilder-teams NPM package, or you can use the bot connector API, which is a bunch of REST APIs allowing you to build the bot in any platform. Imagine that you could write an iOS app where you can say "Hey Siri, ask the Microsoft Bot Framework to do something useful," and then expose that same bot in a Microsoft Team.
3. Develop and test the bot using the bot framework emulator.
4. Deploy the bot to a cloud service. Azure works fine.

Once your bot is written and ready, you need to make it available in teams. You can sideload the bot package and sideload it into a test team for dev purposes. Creating the package involves creating a zip file with:

- A manifest file called manifest.json describing your bot
- A transparent outline icon and a full-color icon meeting certain requirements

Once the bot package is created, you simply sideload it in a team. However, for sideloading to work, you need to enable sideloading of apps. To enable side loading of apps:

1. Sign in as tenant administrator.
2. Under Admin, go to Settings > Services & Add-ins or Apps
3. Find Microsoft Teams in that list.
4. Set the settings as shown in **Figure 3**.

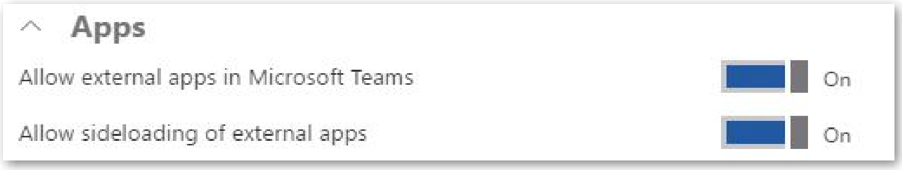

Apps
Allow external apps in Microsoft Teams: On
Allow sideloading of external apps: On

Figure 3: Allowing sideloading of apps

Once you've enabled sideloading, visit the team you wish to sideload using the View Team menu option, as shown in **Figure 4**.

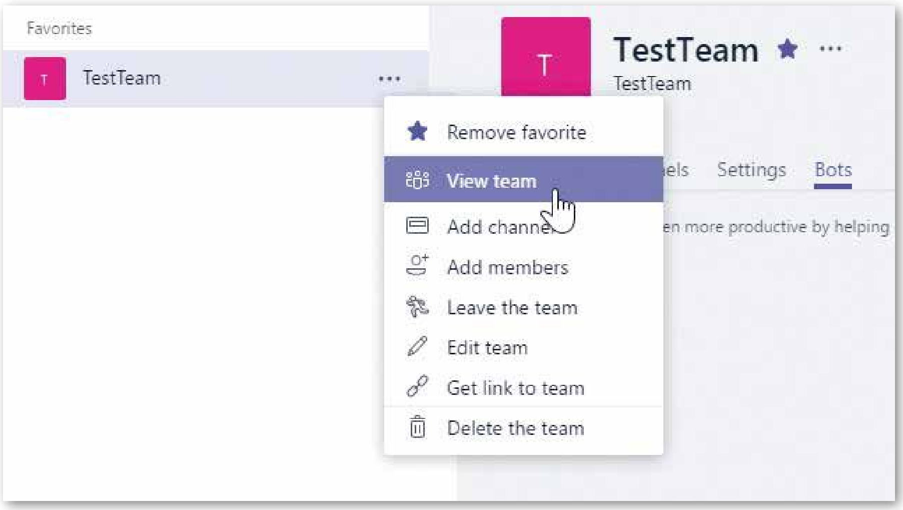


Figure 4: The View team option.

Once on the View Team page, choose **Sideload a bot or tab** option, as shown in **Figure 5.**

Choosing to sideload a bot or tab prompts you to upload the zip file containing the packaged bot. Go ahead and upload it. Once the bot is sideloaded, you can use it by @mentioning it. To access it in direct chats, you can access it either via the App home, or @mention it in a channel.

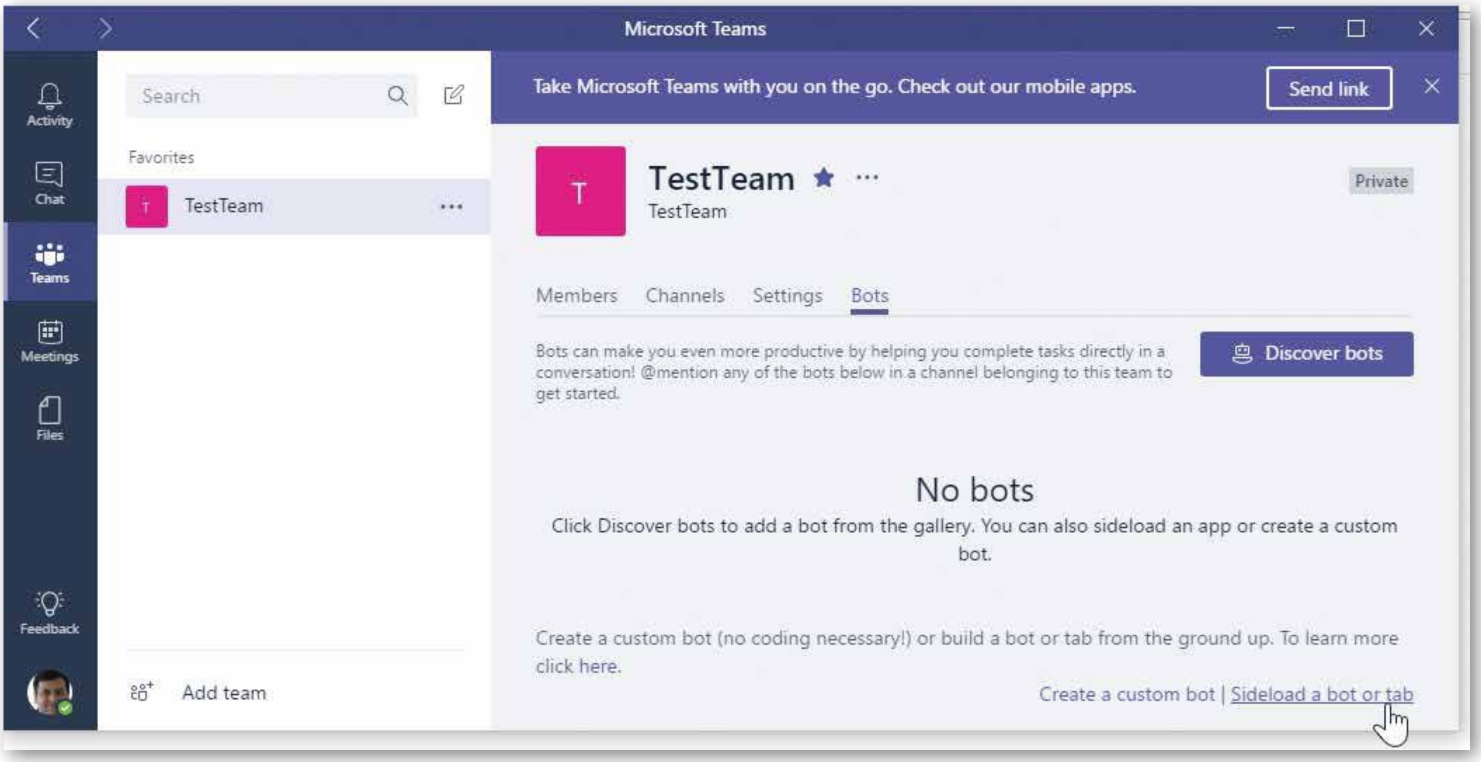
An important thing to remember here is that removing a bot doesn't remove previous conversations. Ideally speaking, you should sideload it into teams as a last step. You can test the bot quite well via the bot-testing framework beforehand.

Connectors

Connectors allow your custom code to push information into teams. I've previously covered Office 365 connectors in depth in the May/June issue of CODE Magazine (http://www.codemag.com/Article/1705031).

Extensions and Sending Messages to the Activity Feed

Extensions allow users to quickly share an app's content


Microsoft Teams

Take Microsoft Teams with you on the go. Check out our mobile apps.

TestTeam

TestTeam

Private

Bots can make you even more productive by helping you complete tasks directly in a conversation! @mention any of the bots below in a channel belonging to this team to get started.

No bots

Click Discover bots to add a bot from the gallery. You can also sideload an app or create a custom bot.

Create a custom bot (no coding necessary!) or build a bot or tab from the ground up. To learn more click here.

Create a custom bot | Sideload a bot or tab

Figure 5: Sideload a bot or tab

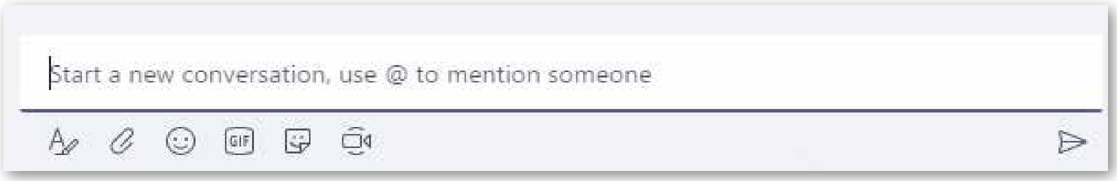


Figure 6: Existing extensions

directly inside a team conversation. Imagine throwing in a bug report as a chat item. Better yet, imagine being able to resolve the bug right within the context of the team. Or being able to interact with a report. The possibilities are endless. These extensions appear as rich cards within the chat. In fact, when you're typing in the chat box, there are some extensions already there at the bottom of the chat window, as can be seen in **Figure 6**.

Adding a custom extension is a matter of authoring a cloud-hosted service that listens to user requests and responds with structured data, such as cards. You can integrate the service with teams via the Bot Framework activity objects. The process of adding an extension is quite simple

Sending messages to the activity feed leverages the Bot Framework. You can flag specific messages so they appear in the user's activity feed as notifications. All you have to do is mark your bot message with the following snippet:

```
"channelData": {
  "notification": {
    "alert": true
  }
}
```

Note that both extensions and sending messages to the user's activity feed are currently in preview.

Summary

SharePoint, the original collaboration tool, is often mistaken as the only collaboration tool Microsoft offers, perhaps because of its widespread use and success. But let's be honest, the dev story of SharePoint has always been clunky. The platform itself is heavy and mired in decisions taken in 2001 or before. I have no doubt that SharePoint will continue to survive and improve, but improve it must.

Meanwhile, we have Microsoft Teams. It's not a replacement for SharePoint, but think of it like a persistent chatroom that's secure, extensible, compelling, and cross-platform. Extensible is where you, the developer, come in. The possibilities are endless, and I hope to explore them further in future articles.

Until then, be amazing and write some code!

Sahil Malik

Legal Notes: Should Software Developers Be Subject to Professional Standards of Ethical Conduct?

Be careful what you ask for. The topic goes dormant for a while and then like clockwork, the drums bang again for the need for software developers to be subject to professional standards. Often, the topic is presented in terms of craftsmanship and engineering. Engineering is an interesting term because states license Professional Engineers (PEs). These are the

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.

folks who sign off on blue prints for roads, bridges, buildings, etc. PEs typically have an educational requirement from an accredited institution and must pass a rigorous examination. In addition, PEs must take a certain number of continuing education credits on an annual or bi-annual basis.

The same is true for the other licensed professions like law, medicine, and accounting. In addition, members of these professions, by the license they hold, are also subject to the rules of professional responsibility that govern a practice, whether it's law, medicine, or accounting. Further, such professionals are **required** to carry malpractice insurance. Ask yourself now: How many software developers would actually meet such standards, assuming state licensure was required and a suitable standards-making body existed? This article will delve into the rationale for why this topic is continually raised and the potential consequences if software developers were to be subject to professional standards?

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.

Why the Push for Professional Standards?

My take is that the push is about getting respect and that software development is a profession at the same level as other recognized professions. The hard truth is that it isn't recognized as such today because anybody can open a software development practice. There are no generally accepted professional standards. That doesn't mean the attempt hasn't been made. A good example comes from the ACM (The Association for Computing Machinery) where they've published the Software Engineering Code of Ethics and Professional Conduct: http://www.acm.org/about/se-code. Other authors, such as Steve McConnell and Bob Martin, have made attempts in their respective books ***Code Complete*** and ***The Clean Coder***.

You Can Always Be Professional

Setting aside for a moment whether software development is a profession and whether there should be licensure and promulgated standards, don't confuse being in a profession with being professional. If you get paid to write software, you're a professional software developer. Note: That's not a badge of quality or minimum competence. If you get paid for something that could otherwise be considered a hobby or pastime, you're a professional.

The question is, do you act and behave in a professional manner? This is where codes of conduct and ethical considerations come into play. Such codes, like the ones you find at conferences, are an attempt to provide an objective standard of conduct. Put another way, these codes are meant to be objective yardstick to judge behavior. If you violate the code, you can be involuntarily removed from a conference. Lawyers and doctors who violate their codes, which are often referred to as rules of professional conduct, can be sued for malpractice.

Think about that for a moment. What if you, a software developer, could be sued for malpractice? Today, that's an impossibility because there are no objective standards of conduct for software developers. I'll get back to this point in a moment.

The key take-away here is that regardless of the job you hold, you can choose to be professional. That means treating colleagues with respect; which doesn't mean you must always agree with them. Being professional implies that you put the interests of your client and your project before your own interest. At the same time, that doesn't mean you must see a project through, regardless of the cost. Being professional doesn't mean you have make everyone your friend. It's a deep concept that often takes years to master and learn. Those who get it early on tend to be the beneficiaries of good mentorship. It's a learned thing, not something you're born with.

Aren't Certification Exams a Professional Designation?

No, certification exams are most certainly not a professional designation. The same goes for the Microsoft MVP award. Certification exams are tests of minimum competency around a specific product and more specifically, a specific product version. Although it's true that vendor-specific exams can lead to product certification like the MCSD and MCSE (Microsoft Certified Solution Developer and Engineer respectively), that's not the same thing as something like a CFA (Chartered Financial Analyst) or a

ChFC (Chartered Financial Consultant) that are earned in the context of a regulated industry that requires licensure.

Should Software Development Be Regulated?

If you're somebody who believes there should be professional standards and a code of ethics, whether you intended it or not, your answer is that you think software development should be regulated. It's one thing for an individual to adhere to a set of ethical guidelines. It's quite another to have those ethical guidelines mandated to govern conduct.

Standards of professional conduct and ethics are the kinds of things that can't be disclaimed in a contract. For example, in the legal profession, there's Rule 1.8 – Conflict of Interest: Current Clients: Specific Rules. This rule is concerned with making sure that the advice a client receives is unbiased in favor of something that may benefit the attorney or some other client of that attorney. In the software development context, consider the advice you give a client when they ask you if an app should be completely rewritten. What if there's a new technology you want to learn? In this case, you could get paid to learn a new technology. Is that in your client's best interest? The answer is that it depends on the specific facts and circumstances. To what degree has your client given informed consent? To what degree have you disclosed relevant information that would facilitate your client giving informed consent? Regardless of what your contract states, if you violated this rule, you could be sued for malpractice because such a rule cannot be disclaimed in a contract. This is what it really means to be held to standards of professional ethics and liability.

Standards of professional conduct and ethics are the kinds of things that cannot be disclaimed in a contract. Regardless of what your contract states, if you violate a rule, you can be sued for malpractice.

The ACM code of ethics previously referenced deals squarely with conflicts of interest in Principle #4 – **Judgment**, and specifically, rule #4.05: **Disclose to all concerned parties those conflicts of interest that cannot reasonably be avoided or escaped**. In addition to the ACM's code on software engineering, the ACM also has an overarching code of conduct: http://www.acm.org/about-acm/acm-code-of-ethics-and-professional-conduct.

Conclusion

Should software developers be subject to professional standards of ethical conduct? The answer to that question is best answered with another question. Should software development be regulated? The second question matters because without regulation, there can be no cognizable and enforceable professional standards outside of a private contract. If, as a software developer, in a contract you sign you wish to be held to such standards, you're free to write those into the contact. Although your client would likely be thrilled to have a clear yardstick to measure contract performance, I don't know why any business-oriented rational-thinking software developer would voluntarily sign up for additional liability. It's important to separate the altruistic ideal of wanting to do a good job from business and legal liability.

There's also the practical aspect of whether software development can be regulated. You'd need a recognized standards-making body that would likely need some standards for admittance such as experience, exams, etc. The hard question you should ask is how many software developers would be able to measure up.

It's ironic that many software developers refer to themselves as software engineers. It's ironic because engineering is about applying mathematical and scientific principles to problem solving. Most software written today is more about trial and error than anything else. If you want to hold yourself to a higher standard, be professional and conduct yourself accordingly and be accountable. At the end of the day, that's what being a professional is all about.

John V. Petersen

ACM

For more information on the Association for Computing Machinery, consult their website at acm.org. Organizations like the ACM are a valuable resource for professional development. To join as a professional member, you need to hold a Bachelor's degree or have at least two years of employment in information technology.

An Introduction to Native Android and iOS Development with NativeScript

NativeScript is an open-source mobile framework that makes it easier for developers to create stunning cross-platform mobile applications that share a single set of code. How NativeScript does this and why it's better over alternative mobile frameworks is the subject of this article. Since the invention of the smart phone, there have been mobile applications and a need for mobile application developers. Using a technology like Java or Objective-C, a developer could create an application compatible with either Android or iOS. These applications are fast and beautiful, but they aren't without penalty on the developer or development teams.

Nic Raboy
www.thepolyglotdeveloper.com
www.twitter.com/nraboy

Nic Raboy is an advocate of modern Web and mobile development technologies. He has experience in Java, JavaScript, Golang and a variety of frameworks such as Angular, NativeScript, and Apache Cordova. Nic writes about his development experiences related to making Web and mobile development easier to understand.

Common Pitfalls of Mobile Application Development

A major pitfall in mobile application development is the need to learn a different language for each mobile platform.

Take, for example, the following Android code written in Java to make HTTP requests against some remote Web service:

```
RequestQueue queue =
    Volley.newRequestQueue(this);
String url ="https://thepolyglotdeveloper.com";

StringRequest stringRequest =
    new StringRequest(Request.Method.GET, url,
    new Response.Listener<String>() {
        @Override
        public void onResponse(String response) {
            System.out.println(response);
        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(
    VolleyError error
    ) {
            System.out.println("Request Failed!")
        }
    });
queue.add(stringRequest);
```

Much of the code in that snippet was taken from the Android SDK documentation found at https://developer.android.com/training/volley/simple.html.

To accomplish the same request within an iOS application, the Objective-C code might look something like the following:

```
NSURL *url =
    [NSURL URLWithString:
    @"https://thepolyglotdeveloper.com"];
NSURLRequest *request =
    [NSURLRequest requestWithURL:url];
[NSURLConnection sendAsynchronousRequest:request
    queue:[NSOperationQueue mainQueue]
    completionHandler:^(NSURLResponse *response,
        NSData *data, NSError *connectionError) {
            NSString *strData =
    [[NSString alloc]initWithData:data
            NSLog(@"%@", strData)
        }];
```

Much of the code above was taken from https://spring.io/guides/gs/consuming-rest-ios/ to illustrate the development differences between Android and iOS using pure Java and Objective-C.

It's feasible to learn both Java and Objective-C, two very different development technologies. If you're the manager of a development team, it might be fiscally responsible to staff developers for each technology.

The differences in platforms don't end with Java and Objective-C, but extend to how the user interface is developed using very different XML tags and attributes.

Mobile developers have been struggling with these platform differences since the beginning, which spawned ideas to find a better way to get the job done.

XML, CSS, and JavaScript or TypeScript

The Web has had plenty of time to mature. It's the norm now for developers to possess skills in Web development no matter their language expertise, so it's not a leap to needing those skills applied to mobile.

With NativeScript and similar frameworks, like Ionic Framework and React Native, developers can take their knowledge of Web development and apply it toward mobile applications, all with a single codebase.

Looking back at the previous example for making HTTP requests within iOS and Android applications, the following accomplishes the same thing in NativeScript:

```
http.getJSON(
    "https://thepolyglotdeveloper.com"
).then(r => {
    console.dir(r);
}, e => {
    console.dir(e);
});
```

The above snippet was taken from the NativeScript Core documentation found at https://docs.nativescript.org/cookbook/http, but not only is it significantly smaller than the Objective-C and Java equivalents, but it's both Android- and iOs-compatible.

NativeScript supports the development of cross-platform mobile applications with JavaScript or TypeScript, a subset of CSS, and an XML markup not too different from HTML.

The icing on the cake is in the framework support that's available with NativeScript. Although NativeScript Core is an option, frameworks like Angular and Vue.js are also options.

The icing on the cake is in the framework support that's available with NativeScript.

Native APIs and Performance without a Web View

NativeScript isn't the only cross-platform mobile development framework out there that works with common Web technologies or even popular Web frameworks. For example, Apache Cordova, Ionic Framework, and other frameworks are also very popular.

So why choose NativeScript over the alternatives?

Apache Cordova-based frameworks render everything within a WebView component, which is an embedded Web browser, rather than mapping to individual native components. The problem with WebView components is that they have limitations and are known to behave differently on varying hardware and platform operating system versions.

Performance is critical in today's consumer facing mobile applications, and so is functionality.

Not only do NativeScript applications render outside a WebView, but native-platform APIs can be accessed directly from the JavaScript or TypeScript code.

Take, for instance, the following Java snippet for Android:

```
import android.support.design.widget.Snackbar;
Snackbar mySnackbar = Snackbar.make(
    findViewById(R.id.myCoordinatorLayout),
    "Hello World!",
    Snackbar.LENGTH_SHORT
);
mySnackbar.show();
```

That code presents a Snackbar in the UI. The Snackbar, like many Android APIs, isn't conveniently wrapped and ready to go in NativeScript. This doesn't mean that you can't use the Snackbar API, it just means it hasn't been designed to be slick, using as little code as possible.

The next snippet presents a Snackbar in NativeScript. Android and iOS classes are immediately available and accessible via JavaScript and TypeScript in NativeScript applications.

```
var Snackbar =
    android.support.design.widget.Snackbar;
var mySnackbar = Snackbar.make(
    topmost().currentPage.android,
    "Hello World",
    Snackbar.LENGTH_SHORT
);
mySnackbar.show();
```

Accessing the native APIs through JavaScript and TypeScript isn't always the most convenient way to get the job done. The Snackbar example didn't take much to get it working. Let's look at a much more complex example of native APIs, such as fingerprint authentication. This is where plug-ins come into play.

NativeScript has many available third-party plug-ins listed at http://plugins.nativescript.org that take the hard work out of accessing native platform APIs. Take, for example, the fingerprint authentication functionality previously mentioned:

```
fingerprintAuth.
    verifyFingerprintWithCustomFallback({
    message: "Scan Finger",
    fallbackMessage: "Enter PIN"
}).then(() => {
        console.log("Authenticated");
}, error => {
    console.dir(error);
});
```

Through the fingerprint authentication plug-in, http://plugins.nativescript.org/plugin/nativescript-fingerprint-auth, what would have been lengthy Objective-C, Java, or even JavaScript code, was significantly reduced through the third-party library.

To be fair, Apache Cordova-based frameworks have access to native platform APIs through the use of plug-ins. However, these APIs can only be accessed via a defined plug-in, rather than anywhere that JavaScript or TypeScript is supported, like what NativeScript offers.

How It's Possible

NativeScript works by leveraging the V8 JavaScript engine for Android and JavaScriptCore for iOS. V8, as per the documentation found at https://developers.google.com/v8/, implements EMCAScript, and likewise with Java ScriptCore, as per the documentation found at http://trac.webkit.org/wiki/JavaScriptCore.

This allows NativeScript for Android to have nearly identical support to that found in desktop Chrome and NativeScript, and for iOS to have nearly the same support as desktop Safari.

V8 and JavaScriptCore recognize Android and iOS because the NativeScript runtime injects them. Both JavaScript virtual machines have APIs that offer a lot of customization to make this possible.

More information on how the V8 and JavaScriptCore virtual machines come together to make NativeScript possible can be found in the NativeScript documentation at http://developer.telerik.com/featured/nativescript-works/.

Developing Your First Mobile App with NativeScript

Background on NativeScript can only get you so far. Let's learn about NativeScript by trying it.

Let's explore this by creating an application that displays the current currency conversion of Bitcoin to a non-cryptocurrency alternative. It will consume data from a remote Web service, something that would take a considerable amount of work using Objective-C or Java.

Installing NativeScript for Development

NativeScript, like many technologies lately, can be installed through the Node Package Manager (NPM). With Node.js installed, execute the following bit of code to install the NativeScript CLI through NPM:

```
npm install -g nativescript
```

This command installs NativeScript globally on your computer. During the installation process, you may be prompted to install the Android SDK. Depending on your development goals, follow the prompts.

More information on installing the NativeScript CLI, and preparing it for Android and iOS development, can be found in the official NativeScript documentation at https://docs.nativescript.org/start/quick-setup.

Creating a New Project with the NativeScript CLI

To create a new NativeScript project using the CLI, execute the following:

```
tns create bitcoin-project --template angular
```

A project called **bitcoin-project** is created. The **angular** template specifies use of Angular and TypeScript in this project. Another option is **--template typescript**, which creates a NativeScript Core project that uses TypeScript.

The next examples use Angular with TypeScript.

Developing the Logic for the Application with TypeScript

This will be a single page application so all application logic exists in a single TypeScript file. In particular, the project's **app/app.component.ts** file that was created when creating the new project.

Open this file and you'll see a very basic class:

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

@Component({
    selector: "ns-app",
    templateUrl: "app.component.html",
})
export class AppComponent {

}
```

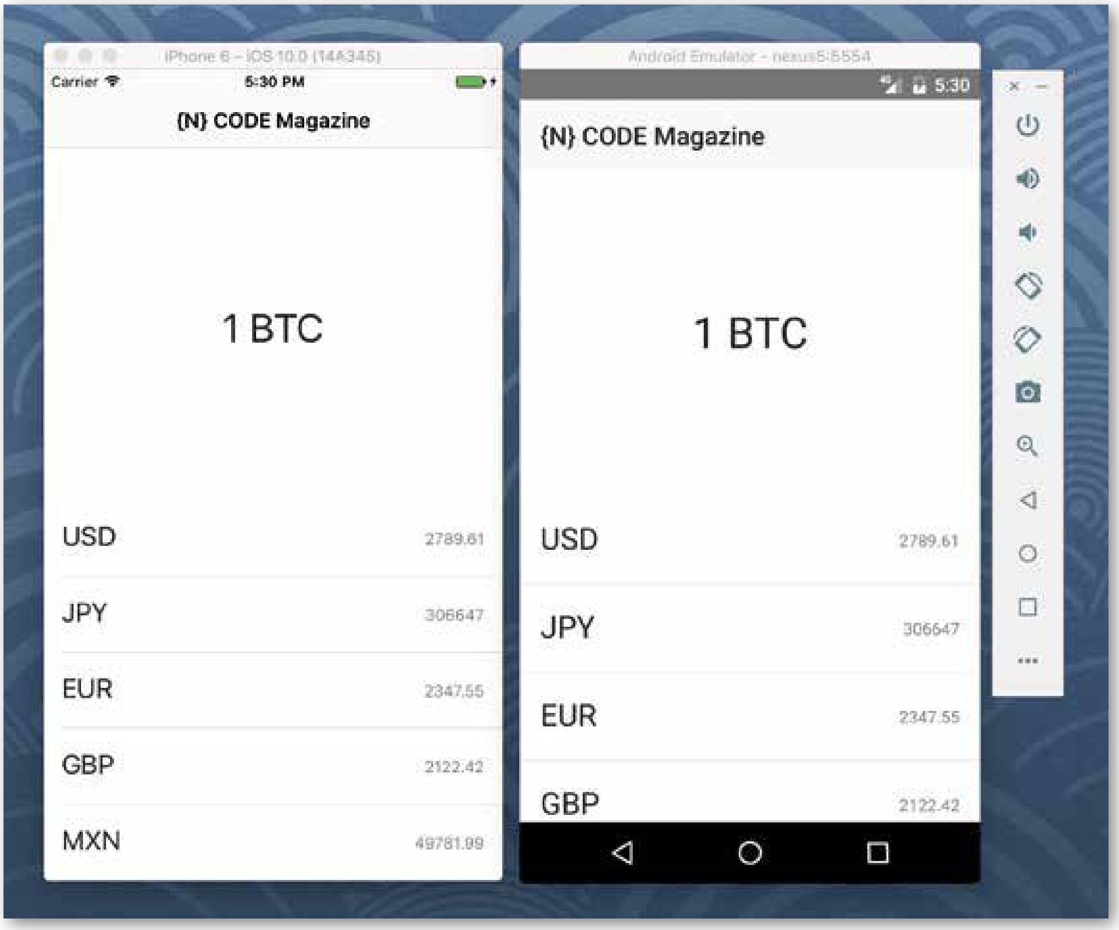


Figure 1: Converting currency app

This TypeScript file is connected to the project's **app/app.component.html** file, which holds the user interface for this particular page.

You need a few more Angular services imported into this page:

```
import {
    Component,
    OnInit
} from "@angular/core";
import { Http } from "@angular/http";
import "rxjs/Rx";
```

The **OnInit** interface gives access to the **ngOnInit** lifecycle event that occurs after the **constructor** executes. The **HTTP** service allows HTTP requests to be made against remote Web services, and **rxjs** allows observables to be used with the HTTP streams.

Because **OnInit** is an interface, the class needs to implement it:

```
export class AppComponent implements OnInit {

    public constructor() { }    public ngOnInit() { }}
```

To prevent compile-time errors, the now implemented interface needs the **ngOnInit** method to be present.

The **constructor** method is used for initializing variables and services and the **ngOnInit** method is used for populating or loading variables using those services. It's a good idea to follow the best practices for using lifecycle hooks, as outlined by the Angular documentation at https://angular.io/guide/lifecycle-hooks.

When it comes to variables in the project, the following two exist with appropriate scopes:

```
private http: Http;
public data: any;
```

The private Http variable is an instance of the imported Http service. The public **data** variable holds any responses received from the Http request. Only public variables can be accessed from the HTML user interface.

A currency type is passed to the API and its value, compared to a single Bitcoin, is returned.

Before Angular services can be used, they must be injected into the **constructor** method:

```
public constructor(http: Http) {
    this.http = http;
    this.data = [];
}
```

To get information about Bitcoin and its currency conversions, the popular Coinbase API is used. Information on the Coinbase API can be found at https://developers.coinbase.com/docs/wallet/guides/price-data. Essentially, a currency type can be passed to the API and its value as compared to a single Bitcoin is returned.

To illustrate a request against Coinbase, a **convert** function can be created:

```
private convert(currency: string) {
    let url = "https://api.coinbase.com";
    url += "/v2/prices/spot?currency=";
    return this.http.get(url + currency)
        .map(result => result.json());
}
```

A currency type such as USD, EUR, etc. can be passed and a GET request is made. The result of the request is transformed using the **map** operator of RxJS and the observable is returned. The request isn't executed until the observable is subscribed to.

Jumping back to the **ngOnInit** method, it might have the following TypeScript:

```
public ngOnInit() {
    let currencies =
    ["USD", "GBP", "EUR", "JPY", "MXN"];
    for(let key in currencies) {
        this.convert(currencies[key])
            .subscribe(result => {
                this.data.push(result.data);
            });
    }
}
```

When the application launches and the **ngOnInit** method triggers, a loop cycles for every currency type in the local **currencies** array. Each currency is sent via HTTP to the Coinbase API and the result is appended to the public **data** array that eventually binds to the user interface.

No other application logic is necessary to power this application.

Designing a Beautiful User Interface with XML

With the page logic ready to go, the corresponding user interface needs to be created. The user interface exists in the project's **app/app.component.html** file as defined in the TypeScript file.

From a design perspective, it needs an action bar, sometimes referred to as a navigation bar. It also needs a list of currencies and a Bitcoin value. To put emphasis on the Bitcoin value, it takes up half the screen.

Open the project's **app/app.component.html** file and include the action bar like this:

```
<ActionBar title="{N} CODE Magazine">
</ActionBar>
```

The page can be split into two sections by using a **GridLayout** with evenly distributed rows and columns.

```
<GridLayout rows="*, *" columns="*">
</GridLayout>
```

The asterisk indicates that a stretch should be made. There are two asterisks defined for rows, meaning that there are two rows. These asterisk values could easily be constant numeric or set to **auto,** which only takes as much height as necessary.

More information on the **GridLayout** and layout containers in general can be found in the NativeScript documentation at https://docs.nativescript.org/ui/layout-containers.

Inside the **GridLayout**, the first row is the Bitcoin value:

```
<Label
    text="1 BTC"
    row="0"
    col="0"
    verticalAlignment="middle"
    class="h1 text-center">
</Label>
```

The text to be displayed is constant and the positioning is constant, starting at the zero index. Various Native Script classes are used to give the text a pleasant appeal. More information on theming in NativeScript can be found in the NativeScript documentation at https://docs.nativescript.org/ui/theme.

The second row of the screen is a **ListView** with dynamic data that comes from the Coinbase API:

```
<ListView
    [items]="data"
    row="1"
    col="0"
    class="list-group">
</ListView>
```

The items of the **ListView** are bound to the public **data** variable found in the TypeScript file.

Each row in the **ListView** needs to be configured. Within the **ListView** tags, the following lines of code exist:

```
<ng-template let-c="item">
</ng-template>
```

Each item in the **data** array will be represented as `C`. For example, to access the Bitcoin amount, **c.amount** is used.

Within each row template, there needs to be another set of columns. This means that there is another **GridLayout**, this time within the **ng-template** tags:

```
<GridLayout
    rows="auto"
    columns="*, *"
    class="list-group-item">
</GridLayout>
```

The inner **GridLayout** has two columns and a row height that only takes as much space as necessary. Each of the two columns are dynamic data:

```
<Label
    text="{{ c.currency }}"
    row="0"
    col="0"
    class="h2">
</Label>
<Label
    text="{{ c.amount }}"
    row="0"
    col="1"
    class="footnote text-right">
</Label>
```

By using curly brackets, the content of the variable is printed rather than the name of the variable.

Although the only page in the single-page application is complete, the base template that came with the new project needs to be altered. This is because the base template came with a little more than what needed to be used. It needs to be altered to prevent run-time errors.

Cleaning the NativeScript Project

The base Angular template for NativeScript ships with some **items** components. These components have routes intended for navigation. Because this is a single page application, neither the components or the routes are required.

Open the project's **app/app.routing.ts** file and empty the **routes** array:

```
const routes: Routes = [];
```

Remove any imports referencing files in the **items** directory.

Now open the project's **app/app.module.ts** file and remove the importing of files in the **items** directory. Also remove references to those files throughout the **@NgModule** block.

Because an Angular service is used in the component that was created, it needs to be imported globally into the project. Within the project's **app/app.module.ts** file, include the following:

```
import { NativeScriptHttpModule } from "nativescript-
angular/http";
```

The **NativeScriptHttpModule** needs to be included in the **imports** array of the **@NgModule** block as well.

Running the Project in the iOS or Android Emulator

At this point, the project is ready to run. In fewer than 40 lines of TypeScript logic and fewer than 15 lines of XML, a cross-platform iOS and Android application is born. To accomplish the same in Objective-C or Java, significantly more would have needed to be done.

To run the emulator, execute the next bit of code from the NativeScript CLI:

```
tns run android --emulator
```

This command assumes that Android was properly configured on the development computer. The **android** argument could easily be replaced with **ios**, provided that Xcode is installed and configured.

Taking the Application to the Next Level with Caching

Many applications that consume remote data need to account for situations where there's no network connection. It's often a good idea to cache data and display it if fresh data can't be obtained.

The Bitcoin conversion rates can easily be stored locally and loaded at will.

To do this, within the project's **app/app.component.ts** file, import the following NativeScript class:

```
import * as ApplicationSettings
    from "application-settings";
```

In fewer than 40 lines of TypeScript logic and fewer than 15 lines of XML, a cross-platform iOS and Android application is born.

The **ApplicationSettings** class allows key-value storage, among other things. The focus here in this article is only on key-value storage.

Within the **AppComponent** class, include the following **save** method:

```
public save() {
    ApplicationSettings.setString(
    "data",
    JSON.stringify(this.data)
    );
}
```

The complex data found in the **data** variable can't be stored in key-value storage, so it must first be serialized into a string.

To load the data, a similar **load** method might exist:

```
public load() {
    this.data =
    JSON.parse(
        ApplicationSettings.getString(
            "data",
            "[]"
        )
    );
}
```

The data found in the **data** key is loaded. If nothing exists at that key, an empty array serialized as a string is loaded. Because the **data** variable is meant to be complex, the serialized data is parsed back into an object.

There are many ways to call these methods, but a common way is from a button in the application action bar. Within the project's **app/app.component.html** file, add the following within the already existing **ActionBar** tags:

```
<ActionItem
    text="Save"
    ios.position="right"
    (tap)="save()">
</ActionItem>
<ActionItem
    text="Load"
    ios.position="left"
    (tap)="load()">
</ActionItem>
```

Running the application yields the same results as the previous live data only version, but this time the caching logic and buttons are available. In this example, the data is not automatically cached and loaded from a local copy.

Conclusion

Developing native mobile applications for Android and iOS no longer has to be difficult or time consuming. With NativeScript, it's very easy to create visually stunning applications with a fraction of the code and time spent during development.

Nic Raboy

CODE

SPONSORED SIDEBAR:

Time to Build a Mobile App?

There's no shortage of options when it comes to languages and platforms for building mobile applications. Android, iOS, or both? Native or hybrid? Which language to choose: Swift, Objective C, Java, C#, JavaScript? Which framework to choose: Xamarin, NativeScript, ReactNative or Cordova?

The experts at CODE Consulting are here to help you decide which option is right for your mobile project. Whether it's a free hour-long mentoring session, coding help to meet an upcoming deadline, or helping you architect and code your mobile app, the experts at CODE Consulting can help with all of your mobile application needs. For more information, visit www.codemag.com/consulting or email us at info@codemag.com and ask about scheduling a free hour-long mentoring session or tell us how our experienced CODE Consultants can help you with your mobile project today.

SQL Server Reporting Services: Eight Power Tips

I'll freely admit, I'm nearly addicted to SQL Server Reporting Services (SSRS). It's not because SSRS is perfect: The product has flaws and shortcomings, like any other product. I'm addicted because I can use SSRS to address many reporting requirements. Although third-party products and self-service reporting tools offer glamorous features beyond what SSRS contains,

Kevin S. Goff
kgoff@kevinsgoff.net
www.KevinSGoff.net
@KevinSGoff

Kevin S. Goff is a Microsoft SQL Server/Data Platform MVP. He's a Database architect/developer/speaker/author, and has been writing for CODE Magazine since 2004. He's 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 to 2015. He creates custom webcasts on SQL/BI topics on his website.

SSRS still provides many capabilities for reporting and dashboarding functionality. I'm fortunate to have had many opportunities to build reporting solutions and will share some of those experiences. In this article, I'll show eight examples that will help you with various reporting tasks.

You've Come a Long Way, SSRS

Twelve years ago, I performed a serious evaluation of SSRS versus Crystal Reports, where the latter was the market leader for reporting products. At the time, I liked some of the features of SSRS, but concluded that it wasn't in the same league as Crystal Reports.

Then Microsoft released SSRS 2008 and I started to warm up to SSRS, as did many other database developers. SSRS 2008 didn't cover every bell and whistle that Crystal Reports contained, but Microsoft enhanced the product enough to make it the proverbial "bread and butter" reporting tool for database developers.

Over the years, I've written four CODE magazine articles that covered SSRS features. Although some of these go back many years, the tips and content still apply to reporting scenarios today. You can find these articles by going to my main author page on the CODE Magazine site (http://www.codemag.com/People/Bio/Kevin.Goff) and searching on the page for SSRS.

Once again, I'll going back to "old faithful" to show eight SSRS tips that have helped me with various report needs.

What's on the Menu?

This article covers a little more than half of a Baker's Dozen this time around, with the following SSRS tips:

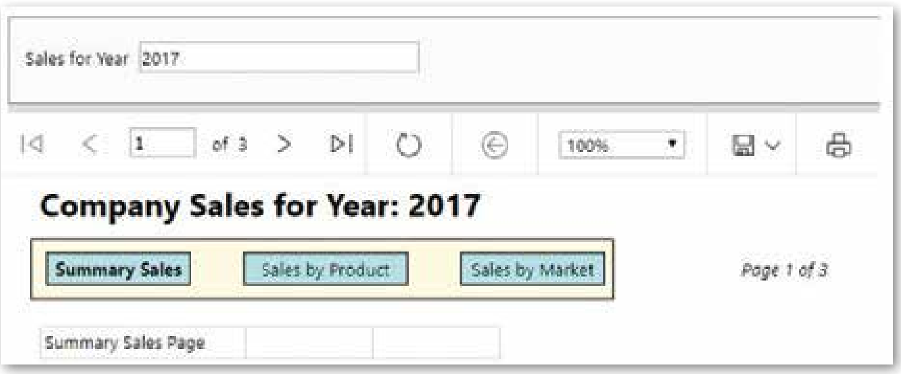
1. Implementing a tab-style interface for navigation to report page/sections
2. Cascading parameters
3. Some tips on Analytic Charts
4. Annotating parameters correctly
5. Scatter charts and drill-down features
6. Multiline tooltips
7. Determining which users have run reports
8. Enhancing document maps for group navigation

Tip #1: Implementing Tab-style Navigation to Report Page/Sections

Almost all SSRS developers have built paginated reports that span dozens or even hundreds of pages. SSRS allows runtime browser navigation to specific pages, either through navigation controls in the browser toolbar or through document map links to specific groupings within the report. In these cases, no one knows until runtime whether the report will be 13 pages, or 33 pages, or more.

Sometimes developers create reports where they intend to show a specific number of pages (usually a handful, at most) as report sections. **Figure 1** shows an example where you might show one page that contains a matrix and a chart for summary sales, a second page for chart breakouts by product, and a third page for breakouts by market. Note the three textboxes in the report page header that serve as navigation controls for the user to quickly access those pages.

Simulating horizontal tabs for navigation is one of many small visual effects that make the output appear nicer.



Company Sales for Year: 2017

Summary Sales | Sales by Product | Sales by Market

Page 1 of 3

Summary Sales Page

Figure 1: An SSRS report with horizontal tabs for page navigation

Yes, you could simply tell the user to navigate to page two or page three using the standard toolbar page navigation controls. But simulating horizontal tabs for navigation is one of many small visual effects that make the output look nicer.

So how can you programmatically jump to a page in SSRS? Some might think that SSRS bookmarks would do the trick, but bookmarks are more for navigation based on some filter/expression or field value. In this case, you've created specific tables or charts in the report that force a page break at the beginning of the object, and you want the user to be able to quickly navigate to that section. The only effective way is to relaunch the report with URL

Listing 1: Query to retrieve filtered years based on Countries selected

```
-- For larger tables, consider using flags in the master tables
-- for whether rows are in distribution
SELECT DISTINCT CalendarYear
FROM            DimDate
WHERE         EXISTS
       (SELECT 1 FROM FactResellerSales
                   INNER JOIN DimReseller
                  ON DimReseller.ResellerKey =
                                                 FactResellerSales.ResellerKey
                                      INNER JOIN DimGeography
                                              ON DimGeography.GeographyKey =
                                                 DimReseller.GeographyKey
                                    WHERE (DimDate.DateKey = FactResellerSales.OrderDateKey)
                                      AND (DimGeography.EnglishCountryRegionName IN
                                          (@prmCountries)))
ORDER BY CalendarYear
```

syntax and programmatically set the Section parameter with the page number. **Figure 2** shows the SSRS action, where you relaunch the report and fill in the SSRS report command for the section number (**rc:Section=2**).

Note that the URL syntax in **Figure 2** includes any values for report parameters. In this case, if the user previously selected "2016" for the report year, pass the current value back into the report when you relaunch it.

When the page reloads, you can also bold the textbox "tab" corresponding to the current page number, by placing the following expression in the font bold property of the textbox:

```
=iif( Globals.PageNumber=2,"Bold","Normal")
```

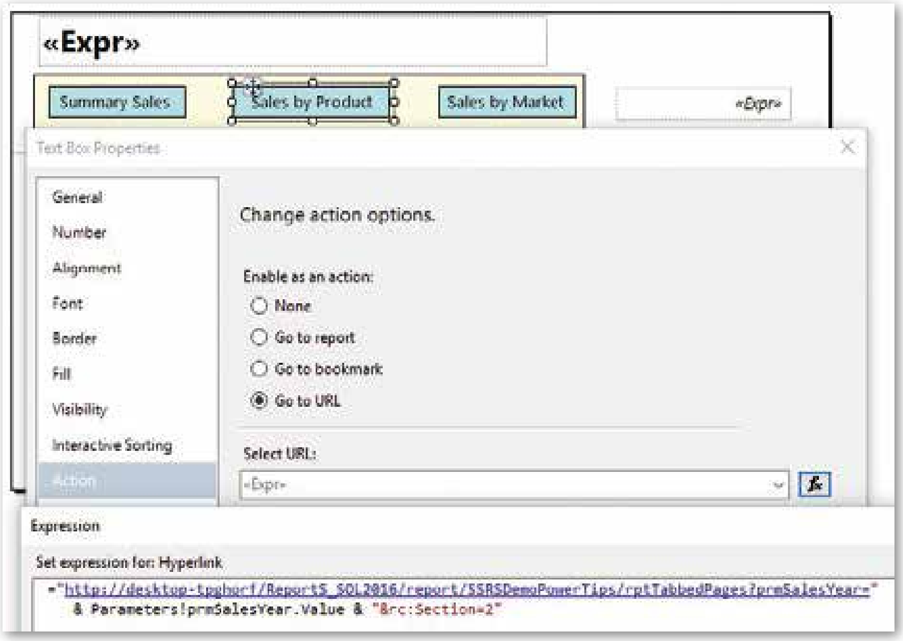
Tip #2: Cascading Parameters

Many years ago, my boss used to leave strongly worded notes on my desk if my software showed drop-down entries that were 100% irrelevant for a particular context. He felt, and rightly so, that drop-down lists should only ever contain relevant values. If a user selected a country, the list of products should only include products with sales in that country; otherwise, the list might contain hundreds of products that weren't relevant for the selection. Yes, there are exceptions—sometimes seeing products without sales might be analytically even more important than products with sales. Still, most of the time you can help users by filtering using just those items that are valid.

Consider **Figure 3** through **Figure 5**, which represent a parameter flow for a sales report. In **Figure 3**, the user selects two countries. In **Figure 4**, the user filters the list of years to only those with sales for France and Germany. (Had I not filtered on years, you'd see several other years in the list). In **Figure 5**, the user filtered the list of products to only those with sales in France and Germany in 2012 and 2013, AND only those products with the word BLUE in the product description.

How do you accomplish this? **Figure 6** shows the "ingredient list": it's three datasets and four parameters. Here's the sequence of events to follow:

1. Create the DataSet **dsCountries**, with a query that reads the list of countries.



```
="http://desktop-tpghorf/ReportS_SQL2016/report/SSRSDemoPowerTips/rptTabbedPages?prmSalesYear="
  & Parameters!prmSalesYear.Value & "&rc:Section=2"
```

Figure 2: Action to relaunch the same report, passing in user selections and the Section number

2. Create the Parameter **prmCountries**, and map the available values of the Parameter to the **dsCountries** Dataset.
3. Create the DataSet **dsYears**, but only pull the years that have sales for the countries that the user selects in the **prmCountries** parameter (**Listing 1**). This provides the "cascading" effect.
4. Create the Parameter **prmYears**, and map the available values of the Parameter to the **dsYears** Dataset.
5. Create the Parameter **prmProductText** as a free-form text parameter (so that the user can enter a text search to further filter the Product list).

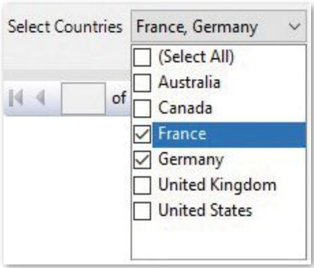


Figure 3: Data-driven drop-down to select Countries

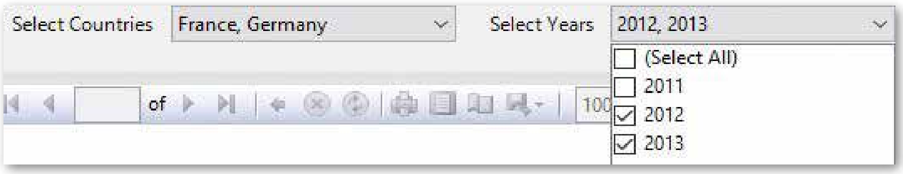


Figure 4: Data-driven drop-down to only show years with sales for selected countries

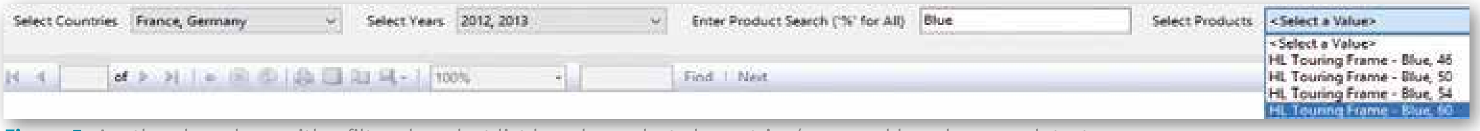

Select Countries
France, Germany
Select Years
2012, 2013
Enter Product Search ('%' for All)
Blue
Select Products
<Select a Value>
<Select a Value>
HL Touring Frame - Blue, 46
HL Touring Frame - Blue, 50
HL Touring Frame - Blue, 54
HL Touring Frame - Blue, 60
of
100%
Find | Next

Figure 5: Another drop-down with a filtered product list based on selected countries/years and based on search text

Listing 2: Query to retrieve filtered products based on Country/Year selected, as well as optional text search

```
-- For larger tables, consider using flags in the master tables
-- for whether rows are in distribution
SELECT ProductKey, EnglishProductName
FROM DimProduct
WHERE EXISTS
 (SELECT       1
    FROM FactResellerSales
            INNER JOIN DimDate
                   ON FactResellerSales.OrderDateKey =
                      DimDate.DateKey
            INNER JOIN DimReseller
                    ON DimReseller.ResellerKey =
                       FactResellerSales.ResellerKey
            INNER JOIN DimGeography
                    ON DimGeography.GeographyKey =
                       DimReseller.GeographyKey
            WHERE  (DimProduct.ProductKey =
                      FactResellerSales.ProductKey) AND
                   (DimGeography.EnglishCountryRegionName IN
                            (@prmCountries)) AND
                   (DimDate.CalendarYear IN (@prmYears))) AND
                      (EnglishProductName LIKE '%' +
                             @prmProductText + '%')
ORDER BY EnglishProductName
```

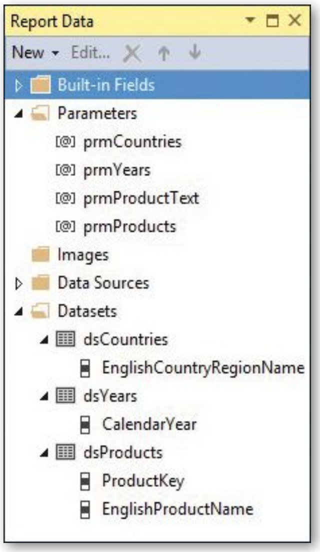

Report Data
New ▾ Edit...
- Built-in Fields
- Parameters
  - prmCountries
  - prmYears
  - prmProductText
  - prmProducts
- Images
- Data Sources
- Datasets
  - dsCountries
    - EnglishCountryRegionName
  - dsYears
    - CalendarYear
  - dsProducts
    - ProductKey
    - EnglishProductName

Figure 6: SSRS Datasets and Parameters required for the cascading exercise

Although this isn't required, it greatly assists the user in narrowing the scope of the product list.

6. Create the Dataset **dsProducts**, but only for the products that have sales for the countries/years that the user selects in the two parameters, and also only for products that contain the text in the **prmProductText** parameter (**Listing 2**).

Use cascading parameters with caution. Not all reports need it. Also, if you're not careful with query strategy, the user might have to wait several seconds—or longer—for SSRS to populate the filtered parameter lists. Still, when you implement properly, this can help guide end users through selections.

Important note: The two queries in **Listings 1** and **2** read from the Microsoft AdventureWorks demo tables, which aren't that large. You wouldn't want to perform these types of queries against very large tables merely to implement cascading parameters. Some environments maintain flags on account and product master tables to mark rows that have been in distribution (i.e., that have sales). That way, any cascading query logic can quickly read these flags instead of large transactional tables to filter parameter lists. The key takeaway is to implement any filtering logic judiciously.

If you're not careful with query strategy, the user might have to wait longer for SSRS to populate filtered parameter lists.

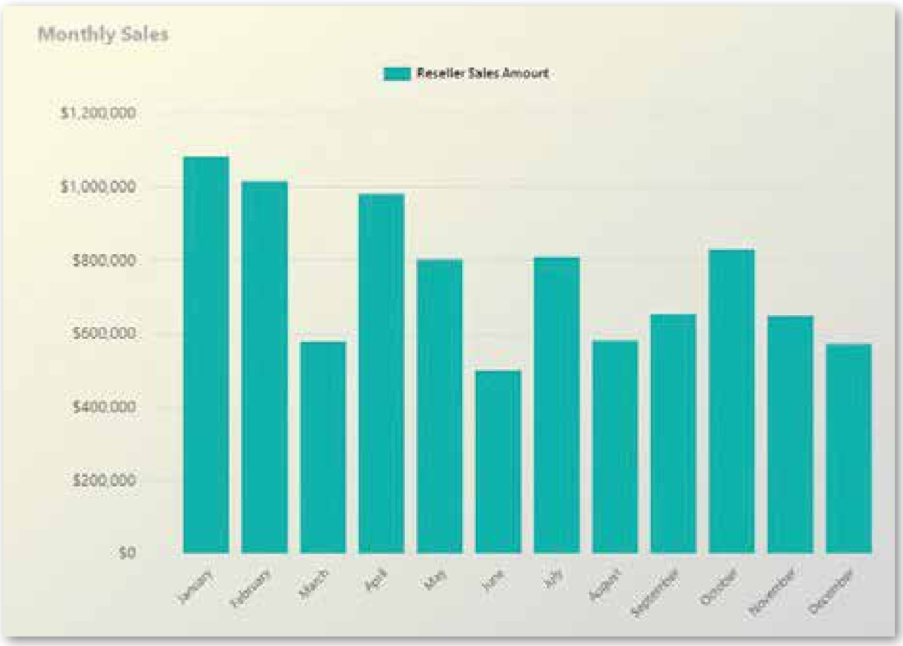


Figure 7: An SSRS bar chart with monthly sales. Can we do better?

Those using SQL Server 2016 with access to the In-Memory Optimized Table feature might also want to consider using In-Memory Tables as part of the strategy for cascading parameter lookup tables. In-Memory Tables offer potentially huge performance boosts, and we all know that users want drop-down lists to populate as quickly as possible!

Tip #3: Some Tips on Analytic Charts

When I was a kid, I'd sometimes ask my father if I could do something that I didn't realize was wrong. My father would respond, "Son, you could do it, but it would be wrong." My corollary is this: Just because you CAN do something, doesn't mean you necessarily should. That is the way I feel about many business charts. Yes, a picture can be worth the proverbial thousand words, but only if it's the right picture. Take **Figure 7** as an example. It's a column chart that shows monthly sales. There's nothing necessarily wrong with this approach, but truthfully, it's rather mundane and a grid of numbers for sales would do just as well.

OK, let's see if we can improve on this. The users are also interested in the average selling price for the current selection. Because the average monthly selling price is on a lower scale than the monthly sales, you need to add a sales price on a secondary Y axis (**Figure 8**).

Figure 8 is certainly an improvement, but you can still do better. You also want to know the overall average price for the year and show which months had an average price above (or below) the overall average price for the year. In **Figure 9**, I add an additional line to show the overall average as a straight red line.

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

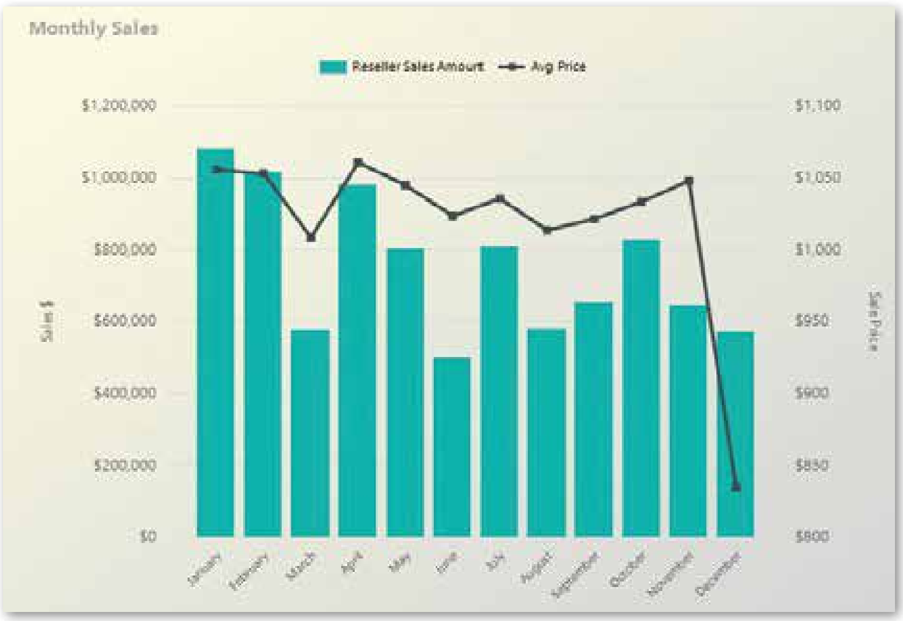


Figure 8: An SSRS monthly sales report with a dual-Y axis to show average sales price. Better, but…

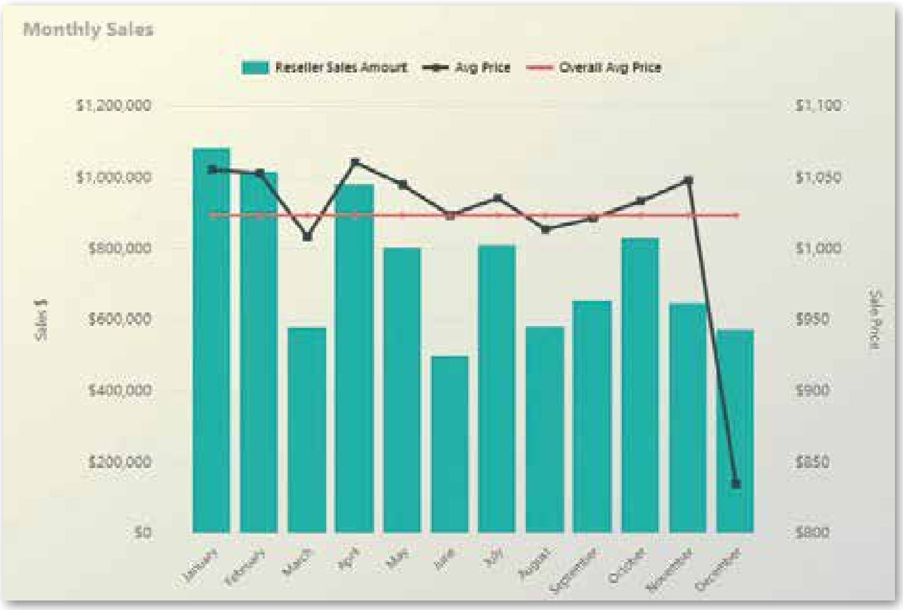
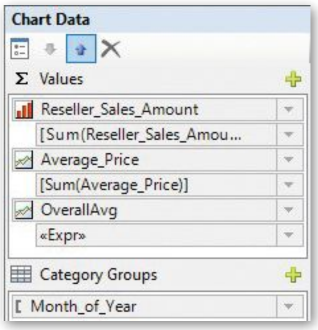


Figure 9: A second horizontal line to show the weighted average over the year. Now we're talking!



Reseller_Sales_Amount
[Sum(Reseller_Sales_Amou...
Average_Price
[Sum(Average_Price)]
OverallAvg
«Expr»
Month_of_Year

Figure 10: The Chart data for the Sales, Average Monthly Price, and expression for Overall Average Price

Figure 10 shows the Chart Data section for the actual chart in **Figure 9**. It plots the reseller sales as a bar chart (note the tiny bar chart image to the left of the reseller sales amount reference). You will also plot the average price and overall average as line charts. SSRS permits you to define a different chart type for each plotted value, which allows you to create the overall chart in **Figure 9**. You can right-click on each specific value to set chart options and configure whether to plot the value on the primary or secondary Y-axis.

Finally, in the overall average value, you can set an expression that (in this instance) calculates an overall average for SSRS to spread as a straight red line. The expression sums the reseller sales across the entire dataset and divides that value by the sum of the order quantity:

```
= sum( Fields!Reseller_Sales_Amount.Value,
          "dsData") /
  sum( Fields!Reseller_Order_Quantity.Value,
          "dsData")
```

The moral of the story is this: Don't build SSRS charts just for the sake of building charts. Every chart should have some compelling message or takeaway.

Don't build SSRS charts just for the sake of building charts!

Tip #4: Annotating Parameters Correctly

Some practices are so obvious that they almost don't require repeating, but I'll do it anyway. Always annotate reports with the user selections. Regardless of whether you show user selections such as Market/Product/Timeframe in a report page heading or footer, always make sure you show them. Imagine if your phone bill didn't show the date range of service or specific accounts or other key information associated with the billing. That's how users can feel if you don't annotate reports with the selections they made!

Tip #5: Scatter Charts and Drill-down Features

One of my favorite chart types is a scatter chart. Recently a company executive asked me to produce a visualization that shows the distribution of price points by customer. They had excess/aged inventory and wanted to sell it to customers who had been paying the highest sales price in specific markets for the same or similar products.

Figure 11 is a bit generic, but still a good example of showing the distribution of data. The example shows the breakout of sales by city for each salesperson, with the average unit price on the Y axis and the sales quantity on the X axis. **Figure 12** shows the chart data components. Admittedly, sometimes I get a little confused about which data element to place in which chart component, so it always helps to keep an example nearby.

Note that **Figure 11** also shows a multi-line tooltip. I'll cover that in the next tip.

Before I continue to the next tip, I'd like to point out a three additional things you can do with scatter charts.

First, some people want to show a straight linear regression trend line to show the impact of the X-axis variable on the Y-axis variable. Unlike Microsoft Excel, SSRS doesn't provide any built-in capability to plot a regression trend line. However, you can find many example SQL queries on blogs/sites to calculate the necessary line slope and plot that data as a straight line.

Second, advanced analysts might want to see the linear correlation between the X-axis and Y-axis variables. Analysts and statistical experts refer to this as the Pearson correlation coefficient, or PCC. In a perfect world where X has a pure linear correlation with Y, the PCC has a value

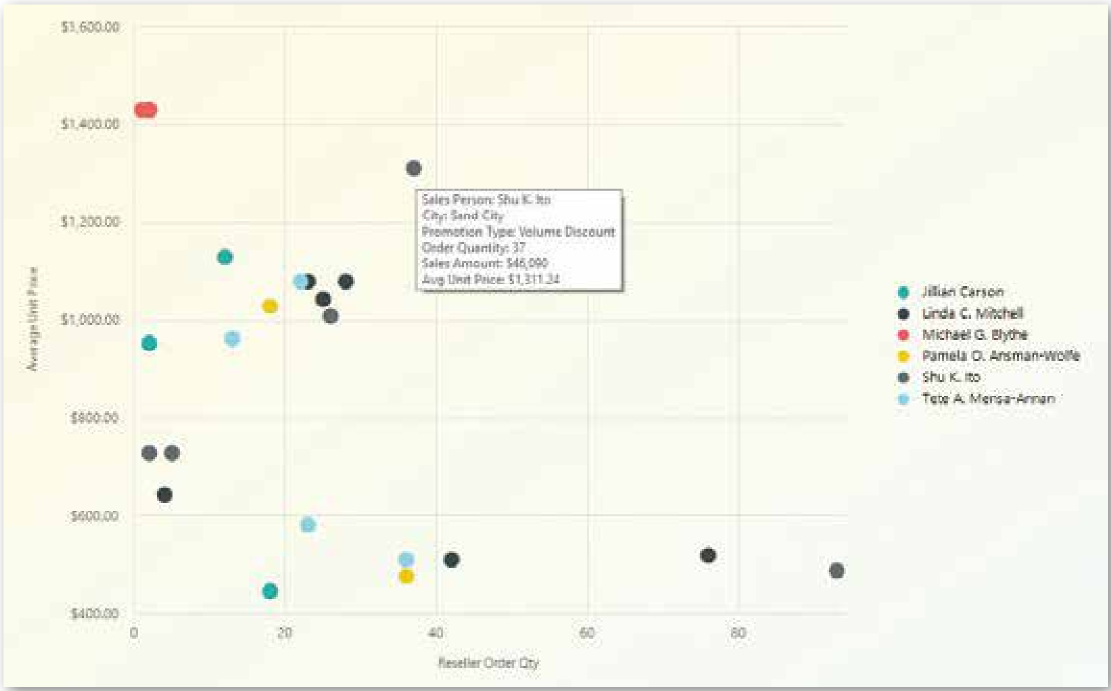


Figure 11: An SSRS scatter chart to show observation points (Employee sales by City)

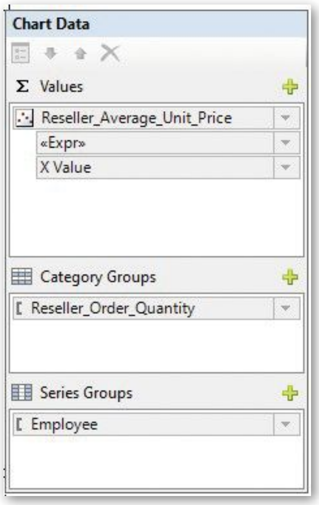


Chart Data

Σ Values

Reseller_Average_Unit_Price

«Expr»

X Value

Category Groups

Reseller_Order_Quantity

Series Groups

Employee

Figure 12: the SSRS Chart data options

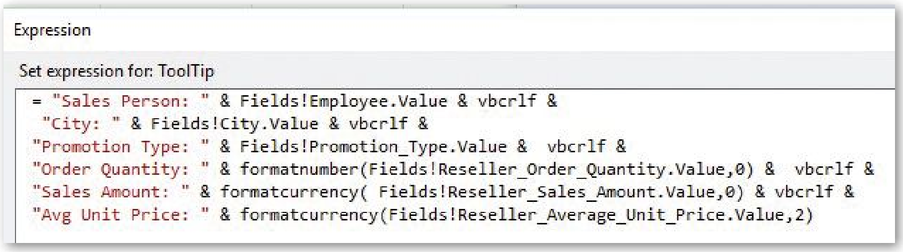
of 1. If X has no linear correlation with Y, the PCC has a value of zero. So overall, the PCC measures the strength of linear dependence between the two variables. Again, analysts use regression lines and PCC values to study the impact of one event on another. In the same way that you can find slope and regression line calculations on blogs and websites, you can also find the logic for the Pearson correlation coefficient.

Third, some people might want to see details associated with a single plotted point on a scatter graph. SSRS makes this very easy through report actions. You can go to the series properties for the data series and define an action based on the data for the current plotted point (i.e., current date, market, product, etc.).

I had a manager who left me screen shots of key on-screen calculations that didn't have an accompanying tooltip, with a VERY angry note. Diplomatic? No. Was he right? Yes.

Tip #6: Multiline Tooltips

Figure 11 in the previous tip shows a multiline tooltip. In my opinion, one of the strongest features in SSRS is how it exposes the current data for the plotted point, making tasks like tooltip expressions and drill-down report actions very easy. In the series data for the chart, you can go to the tooltip property and generate a multiline tooltip (**Figure 13**). Note in the tooltip expression that you use the VBCRLF constant to implement a line break.



Expression

Set expression for: ToolTip

```
= "Sales Person: " & Fields!Employee.Value & vbcrlf &
 "City: " & Fields!City.Value & vbcrlf &
"Promotion Type: " & Fields!Promotion_Type.Value &  vbcrlf &
"Order Quantity: " & formatnumber(Fields!Reseller_Order_Quantity.Value,0) &  vbcrlf &
"Sales Amount: " & formatcurrency( Fields!Reseller_Sales_Amount.Value,0) & vbcrlf &
"Avg Unit Price: " & formatcurrency(Fields!Reseller_Average_Unit_Price.Value,2)
```

Figure 13: An SSRS scatter chart to show observation points (Employee Sales by City)

Use tooltips as much as possible. Users appreciate meaningful information in tooltips! Once, I had a manager who left me screen shots of key on-screen calculations that didn't have an accompanying tooltip, with VERY angry comments. Diplomatic? No. Was he right? Yes.

Tip #7: Determining Which Users Have Run Reports

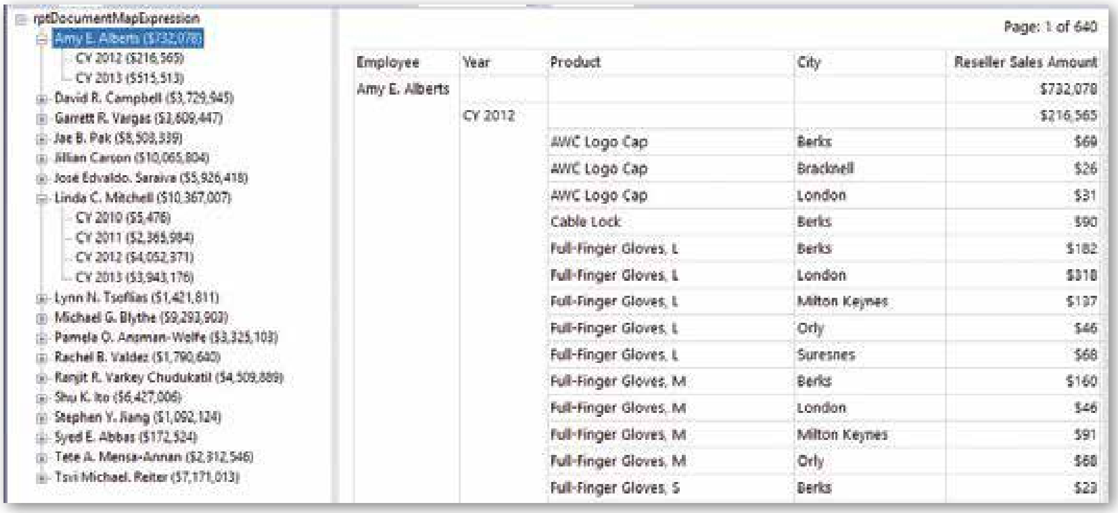
Occasionally, I've needed to see which users have run reports. Fortunately, SSRS provides an execution log in the **ReportServer** database that you can query to see when users have rendered reports.

Here are two SQL queries that you can use against the SSRS Execution Log. The first query lists the reports deployed on the current server along with the most recent execution time and the total number of executions for each report.

```
SELECT ItemPath AS ReportName,
       MAX(CAST(TimeStart AS DATE))
               AS MostRecentReportDate,
                  COUNT(*) as NumExecutions
FROM Executionlog3
WHERE ItemAction = 'Render'
GROUP BY ItemPath
ORDER BY MostRecentReportDate DESC
```

Reporting is Far More than Building Report Layouts.

I've probably said this close to a million times. Report strategies need to consider data access, parameter handling, security (when applicable), rendering to different output formats, calculations, etc. The list goes on and on.



rptDocumentMapExpression
- Amy E. Alberts ($732,078)
  - CY 2012 ($216,565)
  - CY 2013 ($515,513)
- David R. Campbell ($3,729,945)
- Garrett R. Vargas ($3,609,447)
- Jae B. Pak ($8,503,339)
- Jillian Carson ($10,065,804)
- Jose Edvaldo. Saraiva ($5,926,418)
- Linda C. Mitchell ($10,367,007)
  - CY 2010 ($5,476)
  - CY 2011 ($2,365,984)
  - CY 2012 ($4,052,371)
  - CY 2013 ($3,943,176)
- Lynn N. Tsoflias ($1,421,811)
- Michael G. Blythe ($9,293,903)
- Pamela O. Ansman-Wolfe ($3,325,103)
- Rachel B. Valdez ($1,790,640)
- Ranjit R. Varkey Chudukatil ($4,509,889)
- Shu K. Ito ($6,427,006)
- Stephen Y. Jiang ($1,092,124)
- Syed E. Abbas ($172,524)
- Tete A. Mensa-Annan ($2,312,546)
- Tsvi Michael. Reiter ($7,171,013)

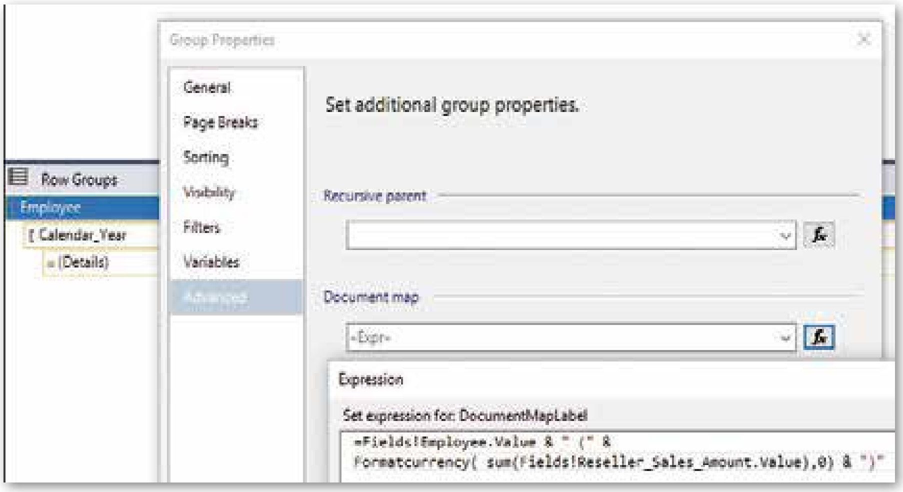
Page: 1 of 640

Employee	Year	Product	City	Reseller Sales Amount
Amy E. Alberts				$732,078
	CY 2012			$216,565
		AWC Logo Cap	Berks	$69
		AWC Logo Cap	Bracknell	$26
		AWC Logo Cap	London	$31
		Cable Lock	Berks	$90
		Full-Finger Gloves, L	Berks	$182
		Full-Finger Gloves, L	London	$318
		Full-Finger Gloves, L	Milton Keynes	$137
		Full-Finger Gloves, L	Orly	$46
		Full-Finger Gloves, L	Suresnes	$68
		Full-Finger Gloves, M	Berks	$160
		Full-Finger Gloves, M	London	$46
		Full-Finger Gloves, M	Milton Keynes	$91
		Full-Finger Gloves, M	Orly	$68
		Full-Finger Gloves, S	Berks	$23

Figure 14: An SSRS report with a Document Map that contains the group names and dollar sales

Use Tooltips!

SSRS provides a decent model for developers to show tooltips with data from the context of the output. Look at a chart or complex calculation on a report and put yourself in the position of the user. What will help the user understand what's behind a particular number?



Set additional group properties.

Recursive parent

Document map: «Expr»

Expression

Set expression for: DocumentMapLabel

```
=Fields!Employee.Value & " (" &
Formatcurrency( sum(Fields!Reseller_Sales_Amount.Value),0) & ")"
```

Figure 15: Document Map properties to set expression for custom Display

The second query shows every user who's run a specific report (referenced in the WHERE clause), along with the most recent execution time and the total number of executions for each user.

```
SELECT  MAX(CAST(TimeStart AS DATE))
              AS MostRecentReportDate,
              COUNT(*) as NumExecutions ,
         UserName
 FROM [ExecutionLog3]
 WHERE
 itempath = '/SomeReport' and ItemAction = 'Render'
 GROUP BY UserName
 ORDER BY MostRecentReportDate DESC
```

I've created administrative-level reports using queries like these in order to gather statistics on report usage.

Tip #8: Enhancing Document Maps for Group Navigation

Most SSRS developers are aware of the Document Map navigation capability, so that users can quickly jump to the start of a specific group value. It's common to see document maps with a list of accounts, products, etc.

However, you can also annotate the document map with additional information, such as the sales for each employee in the group, as well as the sales for the year for that employee (**Figure 14**).

Implementing the document map expression is quite easy. In **Figure 15**, I've pulled up the group properties and set the expression. In this case, the expression concatenates the employee/salesperson with the sum of the reseller sales amount. You can do the same thing for the secondary group on employee plus year (**in Figure 14**) with a similar expression for the year.

Final Thoughts:

I hope that you picked up at least one good tip in this article that you can use in future reports. Although I've largely retired from public speaking, I did roughly 250 community sessions on different .NET and SQL Server topics over the last twelve years and at the end of each session, I asked attendees to raise their hands if they honestly felt they had learned at least one new feature that would help them in some way in their jobs. That has always been my goal. There are many speakers and writers who are far better than I'll ever be. My niche has always been one of a storyteller. I've built many applications for many people and I love sharing how I handled a task in the proverbial trenches.

I bid everyone a temporary *adieu*, as I don my Baker's cap and return the kitchen for ETL in Data Warehousing, part *deux!*

Kevin S. Goff

Every Chart Should Provide at Least One Meaningful Analytic

You could build a column chart that's aesthetically pleasing- and also no more valuable than a regular tabular report. A chart should show something compelling, such as a spike in price relative to an annual average.

Get oodles of free XAML clipart!

MARKUS EGGER'S

XAMALOT

- Get thousands of cliparts free of charge!
- Get clipart in XAML Canvas and XAML Brush formats
- Use live tools to manipulate cliparts
- Styles, skins, templates, and shaders coming soon!
- SVG, JPG, PNG, BMP, GIF, and other formats are also available!

www.Xamalot.com

Digging into Azure Functions: It's Time to Take Them Seriously

From large desktop applications to client-server applications, to the Web, to mobile, and now to AI, software has changed from being centralized to being decentralized. In today's computing landscape, we still have large pieces of software, but smaller, independent components are increasingly common. You may remember batch jobs running as Windows services or EXE applications running as scheduled jobs. Azure Functions combine the best of the worlds of scheduled software and Web services called by another system. Read on to ramp up on:

- Understanding the progression to Azure Functions
- Choosing the right payment model
- Setting the proper Azure configuration
- Developing with triggers, inputs, and outputs

Jeffrey Palermo
jeffrey@clear-measure.com
JeffreyPalermo.com
@JeffreyPalermo

Jeffrey Palermo is the CEO of Clear Measure, a software engineering firm for mid-market non-technology companies. A Microsoft MVP for 11 consecutive years, Jeffrey has spoken at national conferences such as Tech Ed, VS Live, DevTeach, and Ignite. He founded and ran several software user groups and is the author of several books and articles. He's a graduate of Texas A&M University and the Jack Welch Management Institute, an Eagle Scout, and an Iraq war veteran.

Justin Self
justinself@outlook.com
www.justinself.com
@thejustinself

Justin Self works at Tethr, building a voice analytics platform that lets businesses hear what their customers are saying. He speaks at user groups and conferences, and leads the Azure Austin user group.

What Are Azure Functions?

Azure Functions are the next logical step in Platform as a Service, or PaaS. Azure Functions provide the ability to run discrete small units of code, or functions, in an extremely flexible, scalable, and cost-effective manner. Azure Functions offer the ultimate in infrastructure abstraction, removing any concerns about the underlying servers or operating systems. Often dubbed a "Serverless" technology, Functions allow for the quickest path from idea to business value.

Azure Functions combine the best worlds of scheduled software and Web services.

An Evolution

Cloud offerings exist to simplify or eliminate many infrastructure concerns and allow teams to focus on delivering value. Starting with virtual machines (VMs), teams no longer need to be concerned about physical servers. They can customize the environment and create applications without thinking about what happens if a hard drive crashes. When PaaS was introduced, products like Azure App Services removed another layer of concern for developers. Developers no longer needed to care about the operating system! Using PaaS, your team can simply create an application and give it to Azure for hosting. Functions are the next evolution of PaaS.

Now, instead of needing to spin up an entire ASP.NET MVC application with controllers, routes, configurations, build scripts, and deploy scripts to test out an idea, you can simply provision a new Function and go from idea to deployed proof-of-concept in minutes. With Functions, you can have a production-grade integration or API created and deployed in under 60 seconds. Oh, and you haven't paid any money yet, either.

Traditional cloud pricing models operate on reserving resources and charging you for them even if they aren't being used. Functions completely change that by only charging for the time they're being used. A deployed function incurs no runtime costs if nothing triggers it. Teams can have, literally, hundreds of Functions deployed in a live production environment, complete with enterprise-grade logging, security, and scalability, and without so much as a dime being billed against their account. On a side note, if you manage to have that many functions in production, it's probably time to reconsider your architecture.

Be Viral Ready

One problem with using VMs and standard PaaS offerings is the difficulty in using every hertz of the CPU or kilobyte of RAM efficiently. You can't provision just the right amount of resources all the time because you need to be ready for a spike. Sure, there are many ways of mitigating unnatural load against your servers and APIs, but those ways can be complicated and don't offer any more precision than simply adding another VM or App Service Container.

Functions, however, are built to be scalable by default. The runtime monitors the various ways that a Function can be invoked (known aptly as triggers) and provisions additional instances of the Functions automatically, as needed. Once the load has lowered, the runtime then deprovisions the additional instances. This means that you never need to over provision for fear of not meeting viral demands. Even if the load has tremendous peaks and very low valleys, the Functions runtime expands and contracts automatically to continue to serve requests.

Consumption Plan

Azure provides two hosting models for Functions. The first is called the **Consumption Plan**. This is the canonical way of using Functions. The Consumption Plan offers the elastic scalability and pay-per-use model that Functions are known for. However, it does come with a few caveats:

- Functions running on a Consumption Plan have a timeout of five minutes. Should the function run longer than five minutes, the runtime may abruptly kill the Function and any data not persisted will be lost. It's possible to extend the timeout to 10 minutes, but the timeout is set to five minutes by default
- Memory usage is limited to 1.5 GB. Remember, this is also shared among all the Functions within the Function App.

- Scaling is handled automatically and transparently based on the back pressure of triggers; the unit of scale is the Function App. When a Function App is scaled, an additional instance was provisioned. How and when the runtime scales in Function Apps is heuristic by nature. For example, if a Function is triggered by a new message in an Azure Service Bus Queue, the runtime monitors the depth of the queue and the age of the oldest message to determine if additional instances should be provisioned. There are unique scaling heuristics for each trigger type.
- Functions "turn off" after idling for a period of time and can incur a startup cost in terms of performance. This performance penalty can be mitigated (more on that below).

The pricing model for the Consumption Plan is completely based on use, not provisioning. The cost of a Function is a combination of a GB-s (a unit of resource consumption) and the number of executions.

The pricing model for the Consumption Plan is completely based on use, not provisioning.

The formula for calculating GB-s is as follows: **GB-s = (number of executions) x (execution duration in seconds) x (amount of RAM used in GB).**

Once the GB-s has been calculated, the cost becomes $0.000016 per GB-s + $0.20 per million executions. **Table-1** illustrates an example of a Function executed two million times, taking 500 milliseconds each time and using 512 MB of RAM. Execution times are rounded up to the nearest 100 milliseconds and RAM is rounded to the nearest 128 MB. That means that the minimum amount of resources a Function can use is 100 millisecond executions and 128 MB of RAM.

The minimum amount of resources a Function can use is 100 millisecond executions and 128 MB of RAM.

Azure does offer a monthly grant of 400,000 GB-s and 1 million executions. So, if the Function operates within those constraints, no charge is made for running it.

App Service Plan

The other hosting model is the **App Service Plan**. With the App Service Plan, you select the configuration of a VM to be provisioned. It's the same plan that's used for other PaaS offerings, like Azure Web Apps. The number of cores and amount of RAM is static within the configuration, obviously, so choosing the correct configuration should be thoughtful as the price is directly affected. Here are the main differences between the Consumption Plan and App Service Plan:

Monthly Production Values	
Average Memory Consumption	512 MB
Function Execution Duration	.5 Second
Number of Executions	2,000,000
Calculating Resource Consumption (Time)	
Executions	2,000,000
Multiplied by Execution Duration	X .5 seconds
Resource Consumption (seconds)	1,000,000
Calculating Resource Consumption (Memory)	
Average Memory Used (in GB)	512 MB / 1024 MB = .5 GB
Multiplied by Consumption Seconds	X 1,000,000
Total Resource Consumption (GB-s)	500,000 GB-s
Minus Monthly Free Grant	- 400,000 GB-s
Billable Resource Consumption	100,000 GB-s
Billable Resource Consumption	100,000 GB-s
Multiplied by Cost/GB-s	$0.000016
Resource Consumption Cost	$1.60
Cost of Executions	
Total Executions	2 million
Minus Monthly Free Grant	- 1 million
Billable Executions	1 million
Billable Executions	1 million
Multiplied by $.20 per Million Executions	X $.20
Total Cost of Executions	$0.20
Total Cost of Function	
Resource Consumption Cost	$1.60
Plus Execution Cost	+ $0.20
Total Cost	$1.80

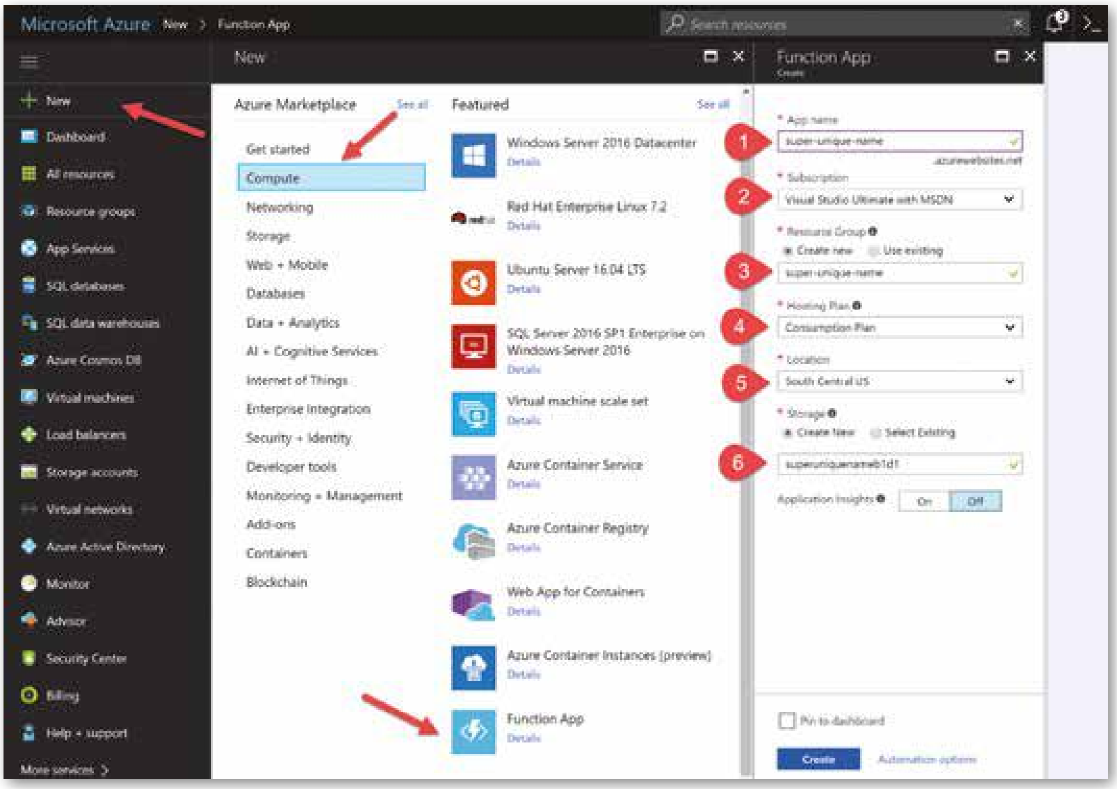
Table 1: An Example of Calculating Costs for a Function

Service	Trigger	Input	Output
Azure Schedule	X		
HTTP (REST or webhook)	X		X
Azure Blob Storage	X	X	X
Event from Azure Event Hubs/Grid	X		X
Azure Storage Queues	X		X
Azure Service Bus Queues and Topics	X		X
Azure Event Grid	X		
External Files *	X	X	X
Azure Storage Tables		X	X
Azure Mobile Apps SQL Tables		X	X
Azure Cosmos DB Documents		X	X
Azure Push Notifications			X
Twilio SMS Text			X
SendGrid Email			X
Bot Framework **			X

*External File triggers and input and output bindings are currently in preview and integrate with File Storage, DropBox, Box, OneDrive, OneDrive for Business, File System, FTP, and SFTP
**Bot Framework output is currently in preview

Table 2: Trigger and Input/Output Bindings Options

- With the App Service Plan, a dedicated VM is provisioned, meaning you are charged for those resources even if they aren't being fully used.
- Scaling beyond the bounds of the VM is not handled by the Functions App runtime and, instead, must be configured manually.
- The memory use of a Function within an App Service Plan is limited to the configuration of the VM. In other words, your limit can be much higher.
- There's no execution time limit. Because the CPU cores are provisioned for your VM, your Function may run for as long as it needs to.



* App name: super-unique-name .azurewebsites.net
* Subscription: Visual Studio Ultimate with MSDN
* Resource Group: Create new / Use existing: super-unique-name
* Hosting Plan: Consumption Plan
* Location: South Central US
* Storage: Create New / Select Existing: superuniquenameb1d1
Application Insights: On [✓ Off]

Figure 1: Creating a new Function App in the Azure Portal

SPONSORED SIDEBAR:

The Dreaded Azure 3 Cs

Confusion, Complication, and Complexity. No doubt about it, Microsoft Azure is a robust and full-featured cloud platform but with that robustness often come the dreaded 3 Cs. CODE Consulting can help you navigate your way through the 3 Cs and partner with you to develop Web, mobile, IoT, SaaS, machine learning (AI) and data analytics solutions on the Microsoft Azure platform. CODE Consulting is your consulting and development partner with the experience and expertise you need. For more information visit www.codemag.com/consulting or email us at info@codemag.com and ask about scheduling a free hour-long mentoring session.

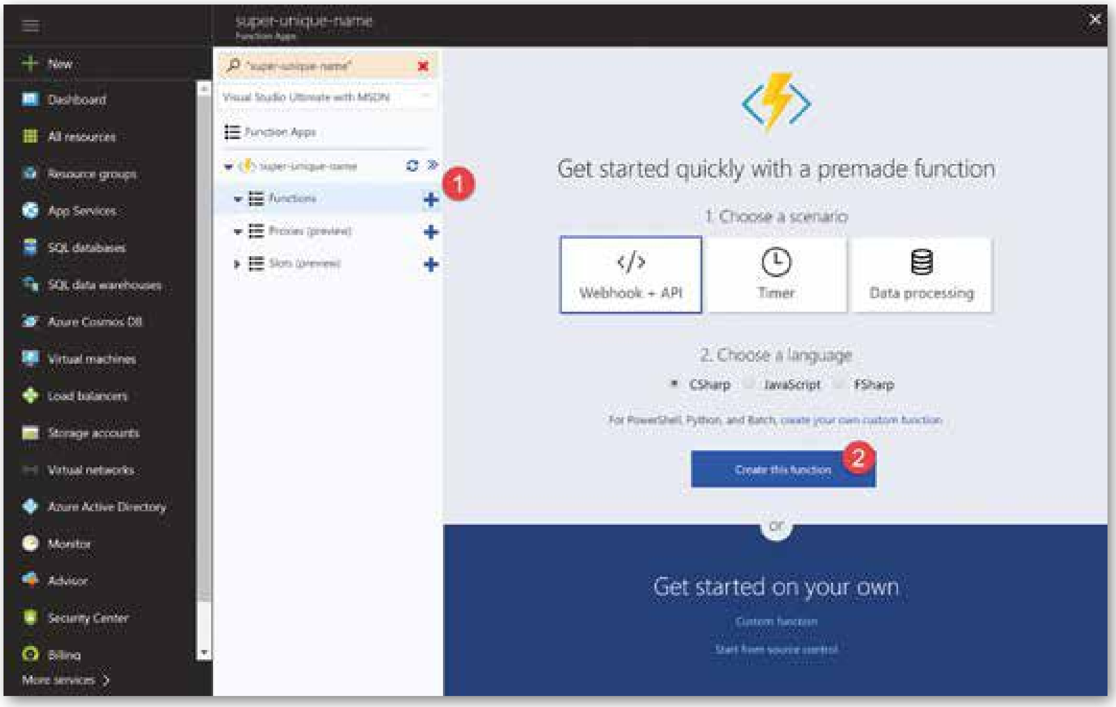
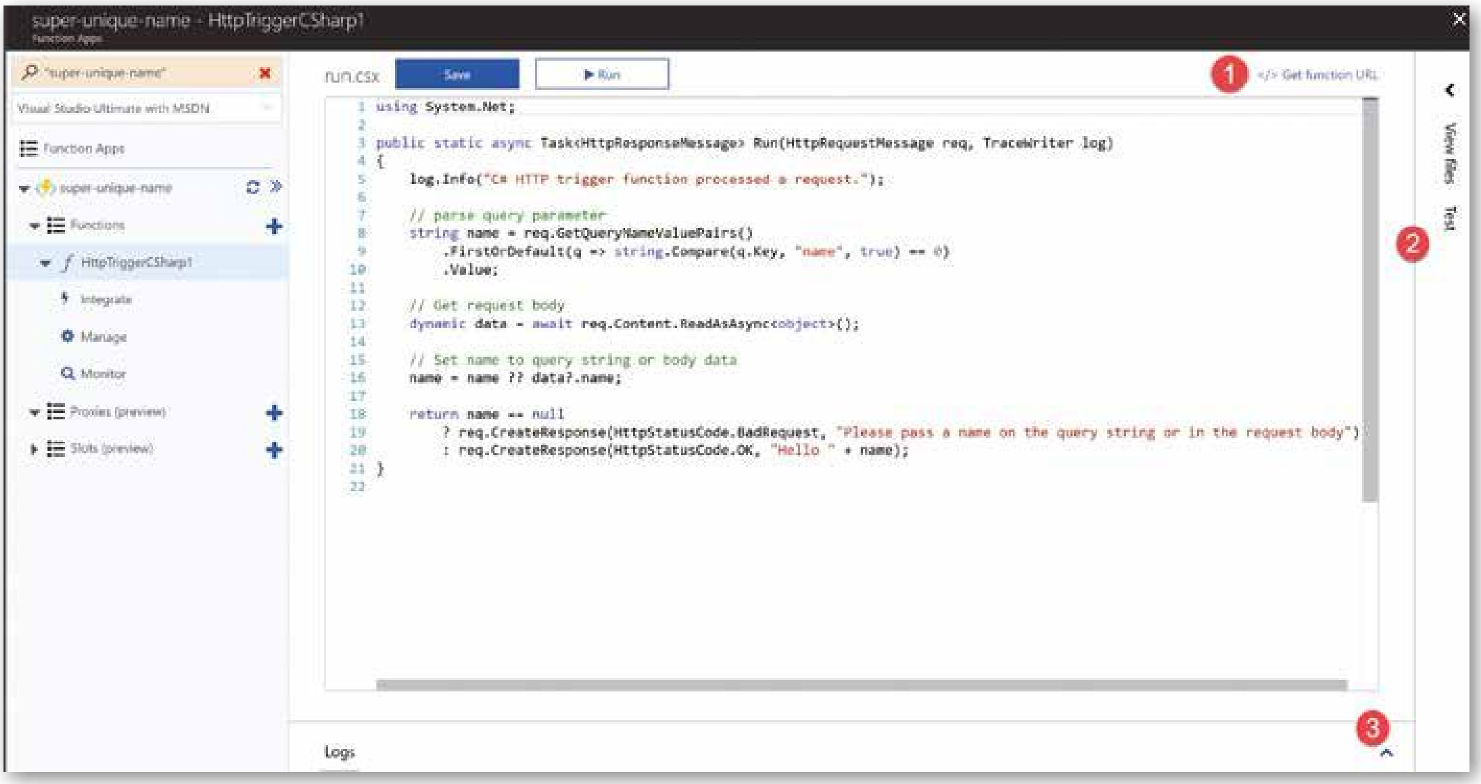


Figure 2: Creating a new Function Using a Quick Template



```
using System.Net;

public static async Task<HttpResponseMessage> Run(HttpRequestMessage req, TraceWriter log)
{
    log.Info("C# HTTP trigger function processed a request.");

    // parse query parameter
    string name = req.GetQueryNameValuePairs()
        .FirstOrDefault(q => string.Compare(q.Key, "name", true) == 0)
        .Value;

    // Get request body
    dynamic data = await req.Content.ReadAsAsync<object>();

    // Set name to query string or body data
    name = name ?? data?.name;

    return name == null
        ? req.CreateResponse(HttpStatusCode.BadRequest, "Please pass a name on the query string or in the request body")
        : req.CreateResponse(HttpStatusCode.OK, "Hello " + name);
}
```

Figure 3: Using the Online Editor

- Functions can be "always on." In App Service Plans, you may configure a Function to be always on, thereby eliminating the idle time, shut off, and accompanying performance penalty that may happen in the Consumption Plan.

Which One to Pick?

The Consumption Plan should be the default choice. However, choose the App Service Plan if:

- You already have an underused App Service Plan that can support your Function app.
- Your Functions consume more than 1.5 GB of memory.
- Your Functions need to run actively for more than 10 minutes.
- The startup performance penalty needs to be eliminated.
- You need to configure Network options for security or access to secured resource (like VNET integration or IP whitelisting).

Triggers, Inputs, and Outputs

All Functions are triggered by some event. It may be a message being added to an Azure Storage Queue, or a new Blob created in Azure Blob Storage. Triggers can also be HTTP requests (either REST or webhooks).

In addition to triggers, Functions can have data bindings for inputting and outputting data. These bindings serve as an easy way to access different resources. The bindings handle connecting to their respective resources and manage the credentials for you. **Table 2** lists the various triggers and the input and output bindings. This list will change as Azure adds capabilities.

The Hello World of Functions

To create a Function, log into the Azure portal. Click **New** on the top left, click **Compute**, then select **Function App** (a Function App is a container for Functions).

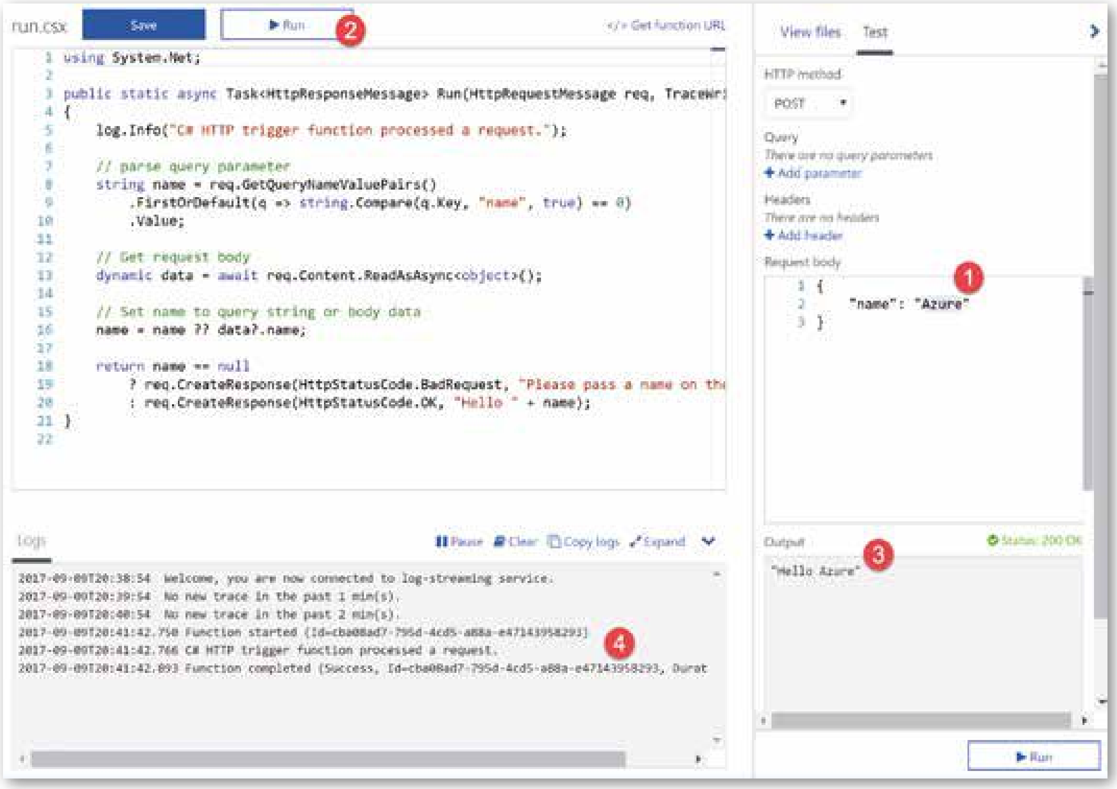
Figure 1 shows the basic configurations you need to make.

1. Give the Function App a globally unique name. It needs to be globally unique because these can be triggered via HTTP requests. Notice the **.azurewebsites.net** that appears underneath the App Name input box at the top of the far right column.
2. Select the appropriate subscription.
3. Use an existing Resource Group or create a new one.
4. Choose the right hosting plan for the Function App: either Consumption or App Service Plan.
5. Select the correct region. As always, keep your resources close together to reduce latency. Calls within the same datacenter experience a latency of around 1-2 milliseconds in overhead. Going from East to West regions, for example, can incur more than 50 milliseconds overheard.
6. Choose your storage account.

Once the Function App is created, clicking the plus (**+**) button next to **Functions** gives an option for quickly creating a new Function using either Webhook or API, and Timer or Data processing triggers. This is shown in **Figure 2**. These premade Functions, used for getting started quickly, come in C#, F# and JavaScript, but you can create custom Functions using any of the following languages: Bash, Batch, C#, F#, JavaScript, PHP, PowerShell, Python, and TypeScript.

The Online Editor

Once the Function is created, you're presented with the online editor; see **Figure 3** for a quick look. Here, you can write C# (or the language you chose to write the Function in) to create APIs, process events, handle incoming data, etc. The function shown in **Figure 3** uses an HTTP trigger and is ready to be invoked. Near the top right, there's a link for grabbing the URL of the Function. On the right lies a built-in mechanism for testing and the bottom shows the live logs. Expanding both of those sections, as shown in **Figure 4**, completes the lightweight development environment in the portal.



```
using System.Net;

public static async Task<HttpResponseMessage> Run(HttpRequestMessage req, TraceWr
{
    log.Info("C# HTTP trigger function processed a request.");

    // parse query parameter
    string name = req.GetQueryNameValuePairs()
        .FirstOrDefault(q => string.Compare(q.Key, "name", true) == 0)
        .Value;

    // Get request body
    dynamic data = await req.Content.ReadAsAsync<object>();

    // Set name to query string or body data
    name = name ?? data?.name;

    return name == null
        ? req.CreateResponse(HttpStatusCode.BadRequest, "Please pass a name on the
        : req.CreateResponse(HttpStatusCode.OK, "Hello " + name);
}
```

```
{
    "name": "Azure"
}
```

```
"Hello Azure"
```

```
2017-09-09T20:38:54  Welcome, you are now connected to log-streaming service.
2017-09-09T20:39:54  No new trace in the past 1 min(s).
2017-09-09T20:40:54  No new trace in the past 2 min(s).
2017-09-09T20:41:42.750 Function started (Id=cba08ad7-795d-4cd5-a88a-e47143958293)
2017-09-09T20:41:42.766 C# HTTP trigger function processed a request.
2017-09-09T20:41:42.893 Function completed (Success, Id=cba08ad7-795d-4cd5-a88a-e47143958293, Durat
```

Figure 4: Using the Online Editor with Test and Logging expanded

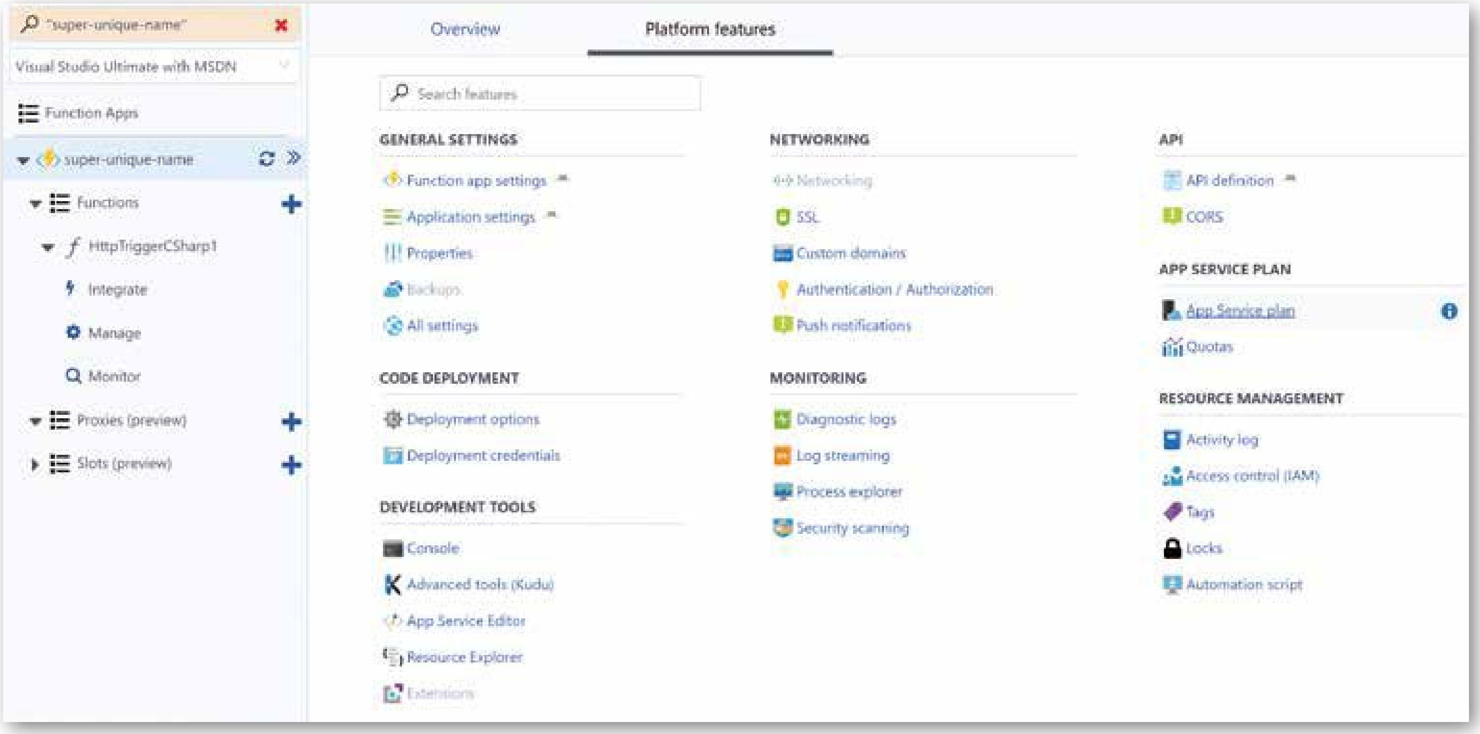


Figure 5: Using the Online Editor with Test and Logging expanded

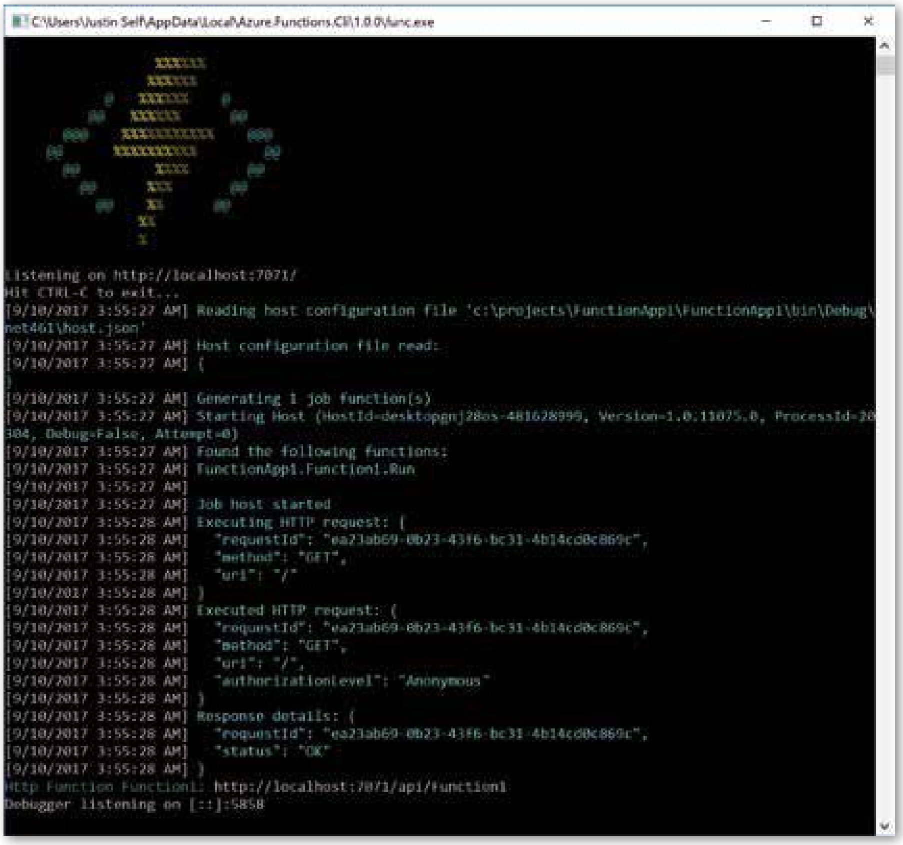
With the Test and Log sections, you'll find a sample body for the HTTP request that comes with the templated Function.

The sample input defaults to setting the name parameter to **Azure**. Here, you can add any parameters that are needed to test the logic in your Function. Clicking Run executes the Function with the test data you've specified. The output is displayed in the bottom right corner and the Logs section shows the successful execution of your new Function.

And that's it. That's the Hello World for Functions. However, that's just the start of Functions. Going back to the Function App blade that was created, there's a tab on top called **Platform Features**. See **Figure 5**.

On the Platform Features tab, the rest of the Functions configuration story comes on the scene. Here, you can easily manage app settings and wire up deployments from a slew of providers, including the most popular Git hosting services. You can integrate your App Insights instance with your Functions. You *are* using App Insights, right?

Other features include setting CORS policies, forcing OAuth2 integrations, custom domains, and SSL certificates. Also, if your Function App uses the App Service Hosting Plan, you can configure VNET integration and whitelisting, among other network configurations.



```
C:\Users\Justin Self\AppData\Local\Azure.Functions.Cli\1.0.0\func.exe

Listening on http://localhost:7071/
Hit CTRL-C to exit...
[9/10/2017 3:55:27 AM] Reading host configuration file 'c:\projects\FunctionApp1\FunctionApp1\bin\Debug\
net461\host.json'
[9/10/2017 3:55:27 AM] Host configuration file read:
[9/10/2017 3:55:27 AM] {
}
[9/10/2017 3:55:27 AM] Generating 1 job function(s)
[9/10/2017 3:55:27 AM] Starting Host (HostId=desktopgnj28os-481628999, Version=1.0.11075.0, ProcessId=20
304, Debug=False, Attempt=0)
[9/10/2017 3:55:27 AM] Found the following functions:
[9/10/2017 3:55:27 AM] FunctionApp1.Function1.Run
[9/10/2017 3:55:27 AM]
[9/10/2017 3:55:27 AM] Job host started
[9/10/2017 3:55:28 AM] Executing HTTP request: {
[9/10/2017 3:55:28 AM]   "requestId": "ea23ab69-0b23-43f6-bc31-4b14cd0c869c",
[9/10/2017 3:55:28 AM]   "method": "GET",
[9/10/2017 3:55:28 AM]   "uri": "/"
[9/10/2017 3:55:28 AM] }
[9/10/2017 3:55:28 AM] Executed HTTP request: {
[9/10/2017 3:55:28 AM]   "requestId": "ea23ab69-0b23-43f6-bc31-4b14cd0c869c",
[9/10/2017 3:55:28 AM]   "method": "GET",
[9/10/2017 3:55:28 AM]   "uri": "/",
[9/10/2017 3:55:28 AM]   "authorizationLevel": "Anonymous"
[9/10/2017 3:55:28 AM] }
[9/10/2017 3:55:28 AM] Response details: {
[9/10/2017 3:55:28 AM]   "requestId": "ea23ab69-0b23-43f6-bc31-4b14cd0c869c",
[9/10/2017 3:55:28 AM]   "status": "OK"
[9/10/2017 3:55:28 AM] }
Http Function Function1: http://localhost:7071/api/Function1
Debugger listening on [::]:5858
```

Figure 6: The Local Function Host Runtime Console

With Visual Studio 2017 15.3.3, you can create Functions locally and even use the local runtime to host and test your Functions.

Local Development

Using the latest Visual Studio 2017 updates (as of this writing, the current version is 15.3.3), you can create Functions locally and even use the local runtime to host and test your Functions. To develop locally, simply create a new project using the Azure Function App project template and add a new Azure Function. The templated file using HTTP triggers is the same as the one online. Pressing F5 from there gives you a console app that loads a small Functions hosting environment; **Figure 6** shows this.

With the local development story complete, you can continue with all the best practices, including implementing a full, end-to-end, Continuous Delivery Pipeline complete with automated testing and deployments.

Conclusion

In this article, you've read an overview of Azure Functions. You've explored the pricing models available, and the methods by which they can be integrated into a production environment with triggers, inputs, and outputs. And you've followed along, configuring the necessary parts of Azure in order to develop your own code that can run in this environment. Now that you know what Functions are, you can explore where they can be used in your production scenarios. With Functions excelling in handling unpredictable loads, scaling to meet massive demand, and extending system architecture via events, you can use them in a number of scenarios. Whether you are refactoring a monolithic system into a series of smaller, "right sized" services, or quickly creating proofs of concept, Functions help your teams deliver value for their customers.

Justin Self

Jeffrey Palermo

Developing Cross-Platform Native Apps with a Functional Scripting Language

In the July/August 2016 issue of CODE Magazine, I published an article on how to create your own scripting language and implement it in C#. I called this language CSCS: Customized Scripting in C#. But I didn't mention any practical usage of such a scripting language at the time, even though there were some unexpected applications of it, e.g., in game hacking.

Vassili Kaplan
vassilik@gmail.com
www.iLanguage.ch

Vassili Kaplan is a former Microsoft Lync developer. He's been writing software since the early nineties, studying and working in a few countries, such as Russia, Mexico, the USA, and Switzerland.

He has a Masters in Applied Mathematics with Specialization in Computational Sciences from Purdue University, West Lafayette, Indiana.

In his spare time, he works on the CSCS scripting language and migrates his free iPhone and Android app iLanguage to CSCS. His other hobbies are traveling, biking, badminton, and enjoying a glass of a good red wine.

Since then, Xamarin was acquired by Microsoft and at first Xamarin Studio Community Edition, and later Visual Studio 2017 Community Edition which contained Xamarin, were released for Windows and macOS. Now individual developers and even small companies can develop iOS and Android apps in C# using Xamarin for free (in addition to the Windows Phone apps that they were already able to develop in C#).

There's a choice of using either Xamarin.Forms (in case users don't require platform-specific functionality and are comfortable with using XAML) or platform-specific Xamarin.iOS and Xamarin.Android to write apps with any features that they can get as if they were using iOS Swift/Objective-C or Android Java development.

I want to be buried with
a mobile phone,
just in case I'm not dead.
Amanda Holden

The first step of shortening time-to-market if you develop cross-platform apps, is to use Xamarin. And this is where I saw the next step and an application for the CSCS scripting language: I can extend the CSCS scripting language for mobile development, so creating and placing different widgets will be just one-liners. The scripting language doesn't have to be used exclusively but can be combined with the C# code.

The most time-consuming part, at least for me, has always been the layout, which is implemented differently on iOS and Android. For Android you usually use XML, and for iOS, there's **Auto Layout**, a constraint-based layout system. Both systems permit having conflicts—definitions conflicting with each other—that are solved at runtime (not always obviously and depending on the screen size).

The advantages of using customized scripting in C# for mobile development are:

- The same code is used to create and place a widget on both iOS and Android. Windows Phone can be added easily as well.
- A simpler layout system works exactly the same for Android and iOS and there's no possibility of conflicts.
- You can call the native C# code from inside a CSCS script. You can avoid delays due to marshalling by pre-compilation. You'll see how to do this in this article.
- The end-result is still a native app.
- Debugging time is quicker with CSCS than with C#. When making some modifications in the script, there's no recompilation of the source code. The changes in the XML/Storyboard aren't necessary anymore for changing the layout (and it does take some time recompiling the layout changes unless you really have a "crème de la crème" development computer).
- Because of the proximity of CSCS to the C# code, you can easily modify the existing CSCS functionality or add a new function. For example, it's very easy to add a new widget, and you'll see some examples in this article.
- The differences from Xamarin.Forms are that you don't need to know XAML, and there fewer lines of code. You can also use platform-specific features more easily.
- People with little or no programming experience can easily create the UI using CSCS scripting.

All of the code in this article is available for free at https://github.com/vassilych/mobile. It's also associated with this article on the CODE Magazine website.

To use CSCS for mobile development, you need to download any version of Visual Studio 2017 and enable the Xamarin option there. Then you can use my sample project at https://github.com/vassilych/mobile or the CODE Magazine website, which contains the CSCS compiler in the shared C# code section, and play around with the script file script.cscs there.

A "Hello, World!" in CSCS

Let's start with our "Hello, World!" program for mobile development. Check out the CSCS script in **Listing 1.**

The result of executing this script is shown in **Figure 1**.

Figure 2 shows fragments of the screen on iPhone and Android after typing "Hi there" in the text field and clicking on the "Change me" button.

As you can see, I added a background and three tabs on the fly. In addition, I added the following widgets: a Label (**UILabel** in iOS and **TextView** in Android), a Button (**UIButton** in iOS and **Button** in Android), and a TextEdit (**UITextField** in iOS and **EditText** in Android).

Let's briefly examine the contents of the "Hello, World!" script in **Listing 1**.

Listing 1: A “Hello, world!” program in the CSCS scripting language

```
function changeme_click(sender, arg) {
  SetText(sender, GetText(textChangeme));
  SetText(versionLabel, _VERSION_ + «. Size: « +
          DisplayWidth + «x» + DisplayHeight);
}

AddTab("Learn", "learn.png", "learn2.png");
SetBackground("ch_bg.png");

locLeft = GetLocation("ROOT", "LEFT", "ROOT", "CENTER", 20, 0);
AddButton(locLeft, "buttonChangeme", "Change me", 260, 80);
AddAction(buttonChangeme,  "changeme_click");
SetFontSize(buttonChangeme, 12);

if (_IOS_) {
  hint = «Hello, iPhone user»
} elif (_ANDROID_) {
  hint = «Hello, Android user»
} else {
  hint = «Hello, Windows user»
}

locLeftRight = GetLocation(buttonChangeme, "RIGHT",
                           buttonChangeme, "CENTER", 40, 0);
AddTextEdit(locLeftRight, "textChangeme", hint, 260, 60);
SetFontSize(textChangeme, 12);

locAbove = GetLocation(buttonChangeme, "ALIGN_LEFT",
                       buttonChangeme, "TOP");
AddLabel(locAbove, "versionLabel", "", 360, 60);

AddTab("Quiz", "test.png", "test2.png");
AddTab("Settings", "settings.png", "settings2.png");

SelectTab(0);
```

The AddTab function creates a tab application on the fly and adds the first tab to the app. Its signature is the following:

```
AddTab(TabName, ActiveIcon, <InactiveIcon>);
```

The ActiveIcon is used when the tab is selected, and the InactiveIcon is used otherwise. The InactiveIcon is optional: if it’s not provided, the ActiveIcon is used.

SetBackground(image) is used to set the background of the app. Note that before using this function, the image file must be first added to the Resources folder of the Xamarin.Android and Xamarin.iOS projects. See the accompanying source code download for details.

The next section explains how you add different widgets to the app.

Layout

In order to add a widget, you need to define the application layout. In this section, you’re going to see how the layout is organized in CSCS. There’s no familiar drag-and-drop functionality but in exchange, there’s more control about where you want your widget placed.

Layout in CSCS

For the layout definition, I used a mixture of the iOS and Android approaches. From iOS and the Auto Layout, I applied a rather obvious concept: For the unique widget location, I need to define the relative widget placement horizontally and vertically, and also the widget size. That’s it! I’m not sure why this can be defined multiple times, and inconsistently, in both iOS and Android, leading to conflicts; these conflicts may be resolved with unexpected results at runtime.

For the implementation, I took an approach similar to the concept of the RelativeLayout in Android, but it’s not possible in CSCS to have multiple definitions for a widget (because they may contradict each other). For instance, in Android, you can apply the method ApplyRule() an unlimited number of times when placing a widget.

To create a new widget, first you need to create its location. The location has the information about the widget’s horizontal and vertical placement. It won’t have the widget’s size: That’s provided later, when you add the widget itself. The reason is that potentially the same location can be used by various widgets of differing sizes (for example, one of them can be hidden, and another can be shown, depending on some other runtime conditions).

Also, the location will have an optional parameter of the view (or layout, in the case of Android) for where to place the widget. If the location isn’t provided, the widget is

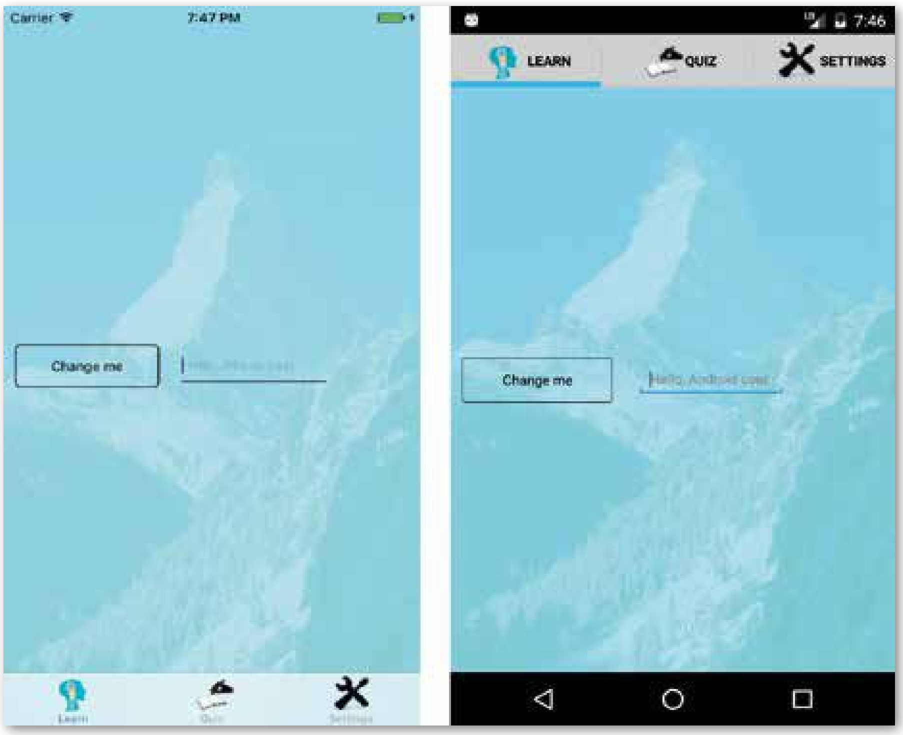


Figure1: Running the “Hello, World!” script on iPhone and Android

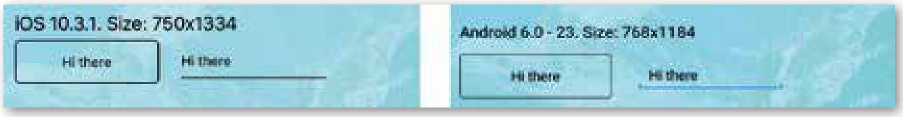


Figure 2: After clicking on the “Change me” button

placed in the root view (or in the root Android layout, which happens to be a RelativeLayout).

To create a widget location, the syntax is the following:

```
Location = GetLocation(horizontalReference,
                       horizontalRelation,
                       verticalReference,
                       verticalRelation,
            <additionalHorizontalMargin>,
            <additionalVerticalMargin>,
            <View>);
```

The horizontal and vertical references are either other widgets (or views) or the root window (in this case, it's denoted by the "ROOT" string). Most of the horizontal and vertical relation parameters are borrowed from the Android RelativeLayout.LayoutParams class. One of the differences is the Center parameter, which is only used for placement inside of the parent in Android, but you use it depending on the context: If the reference widget isn't the root, the new widget is centered relative to the reference widget. You'll see a few examples of placing widgets in different places on the screen and relative to each other later on.

The additional horizontal and vertical margins are, by default, zero. In case they're provided, they indicate the additional margin for moving the widget horizontally (the direction goes from left to right) and vertically (the direction goes from top to bottom). For example, a horizontal margin of -20 means moving the widget left 20 pixels; a vertical margin of 30 means moving the widget down 30 pixels. This is similar to the TranslationX and TranslationY parameters in Android.

Let's see, for example:

```
locLeft = GetLocation("ROOT", "LEFT",
                      "ROOT", "CENTER", 20, 0);
```

The first argument, "ROOT", refers to the reference widget, which is the main window, and the second argument specifies the horizontal placement; in other words, the widget will be placed horizontally on the left. The third and fourth argument specify the vertical placement at the vertical center of the screen. The fifth parameter, 20, indicates that the widget should be moved 20 pixels to the right.

Let's see another example:

```
locAbove = GetLocation(buttonChangeme,
                       "ALIGN_LEFT",
                       buttonChangeme, "TOP");
```

This horizontally aligns the left corner of buttonChangeme with the left corner of the new widget. Vertically, it places the new widget on top of the buttonChangeme.

After creating a location, you must use it to create a widget:

Listing 2: Examples of different layouts in CSCS

```
w        = 220;
h        = 78;
margin   = 10;
fontSize = 14;

AddTab("Learn", "learn.png", "learn2.png");
AddTab("Quiz", "test.png", "test2.png");

locCenter = GetLocation("ROOT", "CENTER", "ROOT", "CENTER");
AddButton(locCenter, "buttonCenter", "", 200, 200);
SetImage(buttonCenter,  "angry.png");

locTrans = GetLocation("ROOT", "CENTER", "buttonCenter", "BOTTOM",
                       0, margin);
AddButton(locTrans, "buttonTrans", "Translate", 200, 80);
AddBorder(buttonTrans, 0, 0);

AddLabel(locTrans, "labelTrans", "", 240, 80);
AlignText(labelTrans, "center");

locCenterLT = GetLocation(buttonCenter, "ALIGN_LEFT", buttonCenter,
                          «TOP», 0, -1 * margin);
AddButton(locCenterLT, "buttonCenterLT", "lt", 85, 85);

locCenterRT = GetLocation(buttonCenter, "ALIGN_RIGHT",
                          buttonCenterLT, «TOP», 0, margin);
AddButton(locCenterRT, "buttonCenterRT", "rt", 85, 85);

locCenterRT2 = GetLocation(buttonCenterRT, "RIGHT", buttonCenterRT,
                           «CENTER», 2, 0);
AddButton(locCenterRT2, "buttonCenterRT2", "rt2", 120, 120);

locCenterRT3 = GetLocation(buttonCenterRT2, "CENTER",
                           buttonCenterRT2, «TOP», 0, -5);
AddButton(locCenterRT3, "buttonCenterRT3", "r", 68, 68);
```

```
locCenterTL = GetLocation(buttonCenter, "LEFT", buttonCenter,
                          «ALIGN_TOP», -1 * margin, 0);
AddButton(locCenterTL, "buttonCenterTL", "tl", 85, 85);

locCenterBR = GetLocation(buttonCenter, "RIGHT", buttonCenter,
                          «ALIGN_BOTTOM», margin, 0);
AddButton(locCenterBR, "buttonCenterBR", "br", 85, 85);

loc1 = GetLocation("ROOT", "LEFT", "ROOT", "BOTTOM");
AddButton(loc1, "button1", "Left", w, h);
SetFontSize(button1, fontSize);

loc2 = GetLocation("button1", "RIGHT", "ROOT", "BOTTOM");
AddButton(loc2, "button2", "Right", w, h);
SetFontSize(button2, fontSize);

loc3 = GetLocation("button2", "RIGHT", "button2", "TOP");
AddButton(loc3, "button3", "RelRight", w, h);
SetFontSize(button3, fontSize);

loc4 = GetLocation("ROOT", "CENTER", "ROOT", "TOP");
AddButton(loc4, "button4", "TopCenter", w, h);
SetFontSize(button4, fontSize);

loc5 = GetLocation("button4", "LEFT", "ROOT", "TOP");
AddButton(loc5, "button5", "TopLeftCenter", w + 10, h);
SetFontSize(button5, fontSize);

loc6 = GetLocation("button4", "RIGHT", "button5", "BOTTOM",
                   -1.5 * w, 0);
AddButton(loc6, "button6", "BelowRight", w, h);
SetFontSize(button6, fontSize);

AddTab("Settings", "settings.png", "settings2.png");
```

Listing 3: A fragment of the UIVariable class

```
public class UIVariable : Variable
{
    public enum UIType { NONE, LOCATION, VIEW, BUTTON, LABEL,
                         TEXT_FIELD, TEXT_VIEW, PICKER_VIEW,
                         IMAGE_VIEW, SWITCH, SLIDER };

    protected static int m_currentTag;

    public UIVariable(UIType type, string name = "",
                      UIVariable refViewX = null,
                      UIVariable refViewY = null)
    {
        WidgetType = type;
        WidgetName = name;
        RefViewX   = refViewX;
        RefViewY   = refViewY;
    }
    public UIType WidgetType     { get; set; }
    public string WidgetName     { get; set; }
    public int Width             { get; set; }
    public int Height            { get; set; }
    public int X                 { get; set; }
    public int Y                 { get; set; }
    public int TranslationX      { get; set; }
    public int TranslationY      { get; set; }

    public string RuleX          { get; set; }
    public string RuleY          { get; set; }

    public UIVariable Location   { get; set; }
    public UIVariable RefViewX   { get; set; }
    public UIVariable RefViewY   { get; set; }
    public UIVariable ParentView { get; set; }
}
```

```
AddButton(locLeft, "buttonChangeme",
          "Change me", 260, 80);
```

This creates a button at the location specified before with the specific width and height in pixels. The button has the title string “Change me”.

The general syntax of a CSCS command to create a widget is the following:

```
AddWidget(widgetType, location, widgetName,
          initializationString, width, height);
```

There are shortcuts for some of the widget types, such as a View, a Button, a Label, etc. The whole list of the functions currently available for the mobile development is shown in **Tables 1, 2,** and **3.** This list is constantly growing, so check the source code at GitHub for up-to-date developments.

The initialization string is context-sensitive. For a label and a button, it sets its text (or its “title” in Android terms). For the TextEdit, it sets a hint (or its “placeholder” in iOS terms). An example of this hint can be seen in **Listing 1**. The initialization string can also provide the image file name for the ImageView and the initialization parameters for some widgets, like Switch and Slider.

I’ve used pixels in this layout; in a future article, you’re going to see how to use DPS (density-independent pixels). Also, even though I’ve given absolute sizes here, it’s easy to make adjustments and have widget sizes and placement depend on the display size because you can use the DisplayWidth and DisplayHeight functions to find out the display size in pixels and use this information to create a multiplication factor for coordinates and sizes.

That’s it, about the layout in CSCS. Now, using the location defined in this section, you can create widgets anywhere on the screen and position them relative to each other.

Example of the Layout in CSCS

Let’s see how to implement the layout shown in **Figure 3** in CSCS.

The example in **Figure 3** uses many different widgets. Check its implementation in **Listing 2**.

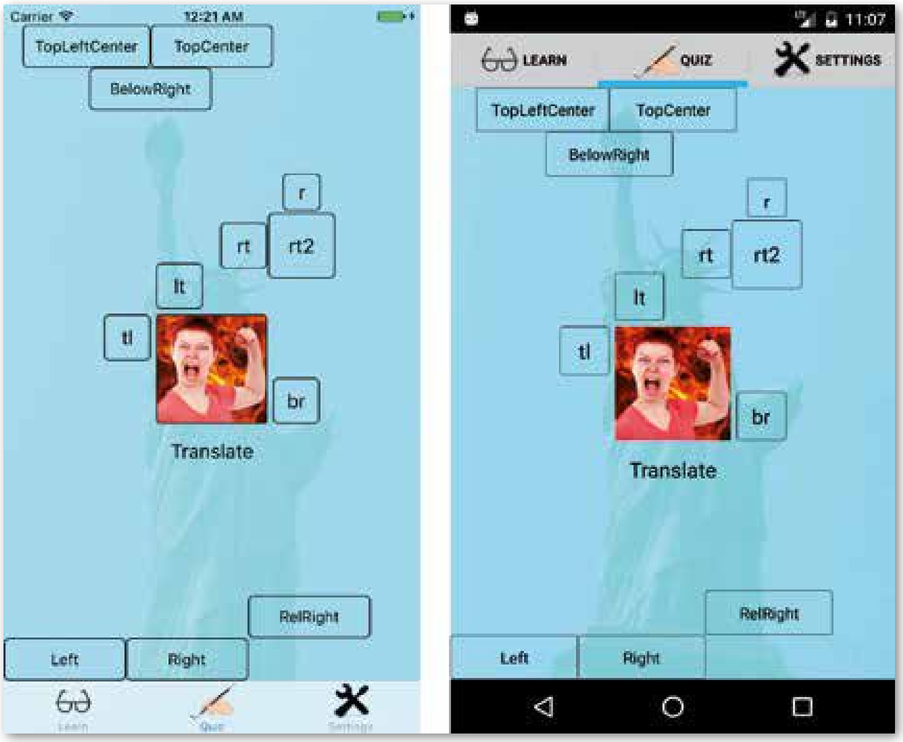


Figure 3: An example of a layout on iOS and Android

Implementation of the Layout in C#

Now let’s see the implementation of the layout in the C# code so you can easily modify it to better fit your needs.

First, I’ll do a very quick elevator pitch of how CSCS scripting language works and how you can add new functions to it. For a longer and a much more detailed explanation, take a look at the article that I published in the July-August 2016 issue of CODE Magazine (http://www.codemag.com/article/1607081).

The CSCS language is based on the Split-and-Merge algorithm, and is implemented in C#. At first, you collect a series of tokens and then merge them one by one. As soon as you encounter an expression in parentheses or a function, you apply the whole Split-and-Merge algorithm to that expression or function. At the second stage of the algorithm, merging, you only have simple expressions, like numbers or strings. And that second merging stage takes into account the priorities of the operators. The first step doesn't take priorities into account.

The CSCS is a functional language where everything turns around functions. Let's see how to add a new function to CSCS. I'll start with simple ones: the functions returning the device's width and height.

The first step is to write a new class deriving from the ParserFunction class, and to override its Evaluate() method. Here's an example for iOS:

```
public class GadgetSizeFunction : ParserFunction
{
  bool m_needWidth;
  public GadgetSizeFunction(
                bool needWidth = true)
```

Listing 4: A fragment of the iOSVariable class

```
public class iOSVariable : UIVariable
{
    public iOSVariable(UIType type, string name,
                       UIView viewx = null, UIView viewy = null) :
                       base(type, name)
    {
        m_viewX = viewx;
        m_viewY = viewy;
        if (type != UIType.LOCATION && m_viewX != null) {
            m_viewX.Tag = ++m_currentTag;
        }
    }
    public CGSize GetParentSize()
    {
        if (ParentView != null) {
            return new CGSize(ParentView.Width, ParentView.Height);
    }
        return UtilsiOS.GetScreenSize();
    }
    public UIView GetParentView()
    {
        iOSVariable parent = ParentView as iOSVariable;
        if (parent != null) {
            return parent.ViewX;
        }
        return AppDelegate.GetCurrentView();
    }

    UIView m_viewX;
    UIView m_viewY;
    string m_originalText;
    string m_alignment;
}
```

Listing 5: A fragment of the DroidVariable class

```
public class DroidVariable : UIVariable
{
    public DroidVariable(UIType type, string name, View viewx,
                         View viewy = null) : base(type, name)
    {
        m_viewX = viewx;
        m_viewY = viewy;
        if (type != UIType.LOCATION && m_viewX != null) {
            m_viewX.Tag = ++m_currentTag;
            m_viewX.Id  = m_currentTag;
        }
    }
    public void SetViewLayout(int width, int height)
    {
        DroidVariable refView = RefViewX as DroidVariable;
        m_viewLayout = MainActivity.CreateViewLayout(width, height,
                       refView?.ViewLayout);
    }
    View        m_viewX;
    View        m_viewY;
    LayoutRules m_layoutRuleX;
    LayoutRules m_layoutRuleY;
    ViewGroup   m_viewLayout; // If this is a parent itself.

    public static Size GetLocation(View view)
    {
        if (view == null) {
            return null;
        }
        int[] outArr = new int[2];
        view.GetLocationOnScreen(outArr);
        return new Size(outArr[0], outArr[1]);
    }
}
```

Listing 6: Getting a Location for Android

```
public class GetLocationFunction : ParserFunction
{
    protected override Variable Evaluate(ParsingScript script)
    {
        bool isList = false;
        List<Variable> args = Utils.GetArgs(script,
             Constants.START_ARG, Constants.END_ARG, out isList);

        string nameX       = args[0].AsString();
        string ruleStrX    = args[1].AsString();
        string nameY       = args[2].AsString();
        string ruleStrY    = args[3].AsString();

        int leftMargin     = Utils.GetSafeInt(args, 4);
        int topMargin      = Utils.GetSafeInt(args, 5);
        Variable parentView = Utils.GetSafeVariable(args, 6, null);

        DroidVariable refViewX = nameX == "ROOT" ? null :
            Utils.GetVariable(nameX, script) as DroidVariable;

        DroidVariable refViewY = nameY == "ROOT" ? null :
            Utils.GetVariable(nameY, script) as DroidVariable;

        DroidVariable location = new DroidVariable(
          UIVariable.UIType.LOCATION, nameX, refViewX, refViewY);

        location.SetRules(ruleStrX, ruleStrY);
        location.ParentView = parentView as DroidVariable;
        location.TranslationX = leftMargin;
        location.TranslationY = topMargin;
        return location;
    }
}
```

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

```
  {
    m_needWidth = needWidth;
  }
  protected override Variable Evaluate(
                   ParsingScript script)
  {
    var nb = UIScreen.MainScreen.NativeBounds;
    return new Variable(m_needWidth ?
              nb.Width : nb.Height);
  }
}
```

As you can see, the function just gets the screen bounds containing both the width and the height, and returns either the width, or the height, depending on the initialization parameter.

Here is the same function implementation for Android:

```
public class GadgetSizeFunction : ParserFunction
{
  bool m_needWidth;
  public GadgetSizeFunction(bool needWidth=true)
  {
    m_needWidth = needWidth;
  }
  protected override Variable Evaluate(
                   ParsingScript script)
  {
    DisplayMetrics dm = new DisplayMetrics();
    MainActivity.TheView.WindowManager.
                DefaultDisplay.GetMetrics(dm);
    return new Variable(m_needWidth ?
            dm.WidthPixels : dm.HeightPixels);
  }
}
```

Listing 7: Adding a Widget. Fragments from the AddWidgetFunction class

```
public class AddWidgetFunction : ParserFunction
{
    public AddWidgetFunction(string widgetType = "")
    {
        m_widgetType = widgetType;
    }
    protected override Variable Evaluate(ParsingScript script)
    {
        string widgetType = m_widgetType;
        int start = string.IsNullOrEmpty(widgetType) ? 1 : 0;
        bool isList = false;
        List<Variable> args = Utils.GetArgs(script,
            Constants.START_ARG, Constants.END_ARG, out isList);

        if (start == 1) {
            widgetType = args[0].AsString();
            Utils.CheckNotEmpty(script, widgetType, m_name);
        }

        iOSVariable location = args[start] as iOSVariable;
        Utils.CheckNotNull(location, m_name);

        double screenRatio = UtilsiOS.GetScreenRatio();

        string varName = args[start + 1].AsString();
        string text    = Utils.GetSafeString(args, start + 2);

        int width      = (int)(Utils.GetSafeInt(args, start + 3) /
                               screenRatio);
        int height     = (int)(Utils.GetSafeInt(args, start + 4) /
                               screenRatio);

        location.SetSize(width, height);
        CGSize parentSize = location.GetParentSize();

        location.X = UtilsiOS.String2Position(location.RuleX,
                     location.ViewX, location, parentSize, true);
        location.Y = UtilsiOS.String2Position(location.RuleY,
                     location.ViewY, location, parentSize, false);

        location.X += location.TranslationX;
        location.Y += location.TranslationY;

        CGRect rect = new CGRect(location.X, location.Y,
                                 width, height);
        iOSVariable widgetFunc = GetWidget(widgetType, varName,
                                           text, rect);
        Utils.CheckNotNull(widgetFunc, m_name);

        var currView = location.GetParentView();
        currView.Add(widgetFunc.ViewX);

        iOSApp.AddView(widgetFunc.ViewX);

        ParserFunction.AddGlobal(varName,
                            new GetVarFunction(widgetFunc));
        return widgetFunc;
    }

    public static iOSVariable GetWidget(string widgetType,
                  string widgetName, string initArg, CGRect rect)
    {
        UIVariable.UIType type = UIVariable.UIType.NONE;
        UIView widget = null;
        switch (widgetType)
        {
            case "Button":
                type = UIVariable.UIType.BUTTON;
                widget = new UIButton(rect);
                ((UIButton)widget).SetTitleColor(UIColor.Black,
                                 UIControlState.Normal);
                ((UIButton)widget).SetTitle(initArg,
                                 UIControlState.Normal);
                AddBorderFunction.AddBorder(widget);
                break;
            case "Label":
                type = UIVariable.UIType.LABEL;
                widget = new UILabel(rect);
                ((UILabel)widget).TextColor = UIColor.Black;
                ((UILabel)widget).Text = initArg;
                break;
            case "TextEdit":
                type = UIVariable.UIType.TEXT_FIELD;
                widget = new UITextField(rect);
                ((UITextField)widget).TextColor = UIColor.Black;
                ((UITextField)widget).Placeholder = initArg;
                MakeBottomBorder(widget);
                break;
            // All other widgets go here...
        }
    }
}
```

Listing 8: Translating a Relation Parameter to a Position on iOS

```
public static int String2Position(string param, UIView
  referenceView, iOSVariable location, CGSize parentSize, bool isX)
{
    bool useRoot = referenceView == null;

    int refX = useRoot ? 0 :  (int)referenceView.Frame.Location.X;
    int refY = useRoot ? 0 :  (int)referenceView.Frame.Location.Y;
    int refWidth = useRoot ?  (int)parentSize.Width :
                              (int)referenceView.Frame.Size.Width;
    int refHeight = useRoot ? (int)parentSize.Height :
                              (int)referenceView.Frame.Size.Height;
    int parentWidth  = (int)parentSize.Width;
    int parentHeight = (int)parentSize.Height;
    int widgetWidth  = (int)location.Width;
    int widgetHeight = (int)location.Height;

    switch (param) {
        case "ALIGN_LEFT": // X
            return useRoot ? 0 :
                             refX;
        case "LEFT": // X
            return useRoot ? 0 :
                             refX - widgetWidth;
        case "ALIGN_RIGHT": // X
            return useRoot ? parentWidth - widgetWidth :
                             refX + refWidth - widgetWidth;
        case "RIGHT": // X
            return useRoot ? parentWidth -  widgetWidth :
                             refX + refWidth;
        case "ALIGN_PARENT_TOP":
        case "ALIGN_TOP": // Y
            return useRoot ? ROOT_TOP_MIN :
                             refY;
        case "TOP":
            return useRoot ? ROOT_TOP_MIN :
                             refY - widgetHeight;
        case "ALIGN_PARENT_BOTTOM":
        case "ALIGN_BOTTOM":
            int offset1 = useRoot ? parentHeight - widgetHeight –
                                    ROOT_BOTTOM_MIN :
                                    refY + refHeight - widgetHeight;
            // if there is a tabbar, move the bottom part up:
            if (useRoot && !isX) {
                offset1 -= (int)(iOSApp.CurrentOffset * 0.8);
            }
            return offset1;
        case "BOTTOM":
            int offset2 = useRoot ? parentHeight - widgetHeight –
                                        ROOT_BOTTOM_MIN :
                                        refY + refHeight;
            // if there is a tabbar, move the bottom part up:
            if (useRoot && !isX) {
                offset2 -= (int)(iOSApp.CurrentOffset * 0.8);
            }
            return offset2;
        case "CENTER":
            if (useRoot) {
              return isX ? (parentWidth  - widgetWidth ) / 2 :
                           (parentHeight - widgetHeight) / 2 ;
            } else {
              return isX ? refX + (refWidth  - widgetWidth ) / 2 :
                         refY + (refHeight - widgetHeight) / 2;
            }
        default:
            return 0;
    }
}
```

Listing 9: Invoking and caching a method using Reflection in C#s

```
using System.Collections.Generic;
using System.Linq.Expressions;
using System.Reflection;

static Dictionary<string, Func<string, string>> m_compiledCode =
   new Dictionary<string, Func<string, string>>();

public static Variable InvokeCall(Type type, string methodName,
      string paramName, string paramValue, object master = null)
{
    string key = type + "_" + methodName + "_" + paramName;
    Func<string, string> func = null;

    // Cache compiled function:
    if (!m_compiledCode.TryGetValue(key, out func)) {
        MethodInfo methodInfo = type.GetMethod(methodName,
                                new Type[] { typeof(string) });
        ParameterExpression param = Expression.Parameter(
                                    typeof(string), paramName);

        MethodCallExpression methodCall = master == null ?
            Expression.Call(methodInfo, param) :
            Expression.Call(Expression.Constant(master),
                            methodInfo, param);
        Expression<Func<string, string>> lambda =
            Expression.Lambda<Func<string, string>>(methodCall,
                              new ParameterExpression[] { param });
        func = lambda.Compile();
        m_compiledCode[key] = func;
    }

    string result = func(paramValue);
    return new Variable(result);
}
```

The second step, after implementing the functions in C#, is to register them with the parser. This registration is now the same for iOS and Android, and it's the following:

```
ParserFunction.RegisterFunction("DisplayWidth",
               new GadgetSizeFunction(true));
ParserFunction.RegisterFunction("DisplayHeight",
               new GadgetSizeFunction(false));
```

This means that as soon as the parser finds the **DisplayWidth** token, the Evaluate() method of the GadgetSizeFunction object initialized with m_needWidth = true is called and as soon as the parser finds the **DisplayHeight** token, the Evaluate() method of the GadgetSizeFunction object initialized with m_needWidth = false is called.

That's it! In my previous article, there was an additional step to register any possible translations supplied in a configuration file, but I'll skip it here for brevity. This is how easy it is to add a new functionality to the CSCS scripting language: Implement an Evaluate method in a new class deriving from the ParserFunction class and then register it with the parser!

The Evaluate method returns an object of the Variable class. This is a generic object used in CSCS scripting. For mobile development, you need a more customized object.

The UIVariable class derives from the variable class and is a wrapper over all of the widgets and locations. A fragment of this class is shown in **Listing 3**. The iOSVariable and DroidVariable are concrete implementations of the UIVariable for the iOS and Android correspondingly. They are shown in **Listing 4** and **Listing 5**.

The first step in a layout operation in CSCS is to get a location for the widget. **Listing 6** shows the implementation of getting a location for Android. The implementation for iOS is very similar.

Once you have a location, you can place the widget there. The code for Android is a bit more straightforward (because the placement is done on a RelativeLayout), so I'll show the code for placing the iOS widgets. The code here is not complete and I encourage you to check out the details in the accompanying source code.

There's one main function for adding a widget to the screen, and which is used for all types of widgets, AddWidgetFunction. There are a few exceptions to this, such as pop-up dialogs, like an Alert Dialog or a Toast, that are implemented differently. In order to use the same function for different widgets, register functions that add widgets, as follows:

```
ParserFunction.RegisterFunction("AddButton",
           new AddWidgetFunction("Button"));
ParserFunction.RegisterFunction("AddLabel",
           new AddWidgetFunction("Label"));
ParserFunction.RegisterFunction("AddTextEdit",
           new AddWidgetFunction("TextEdit"));
```

Continue on in that fashion. **Listing 7** shows the implementation of the AddWidgetFunction class on iOS.

The function responsible for the translation of the ALIGN_LEFT, BOTTOM, CENTER, etc. parameters to the concepts that iOS and Android understand is called String2Position. This function is shown in **Listing 8**. Depending on the layout parameters, it returns a coordinate of the point where you place the widget.

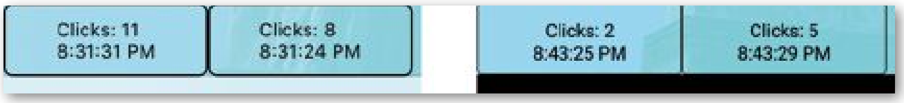


Figure 4: Getting the button title from the C# code

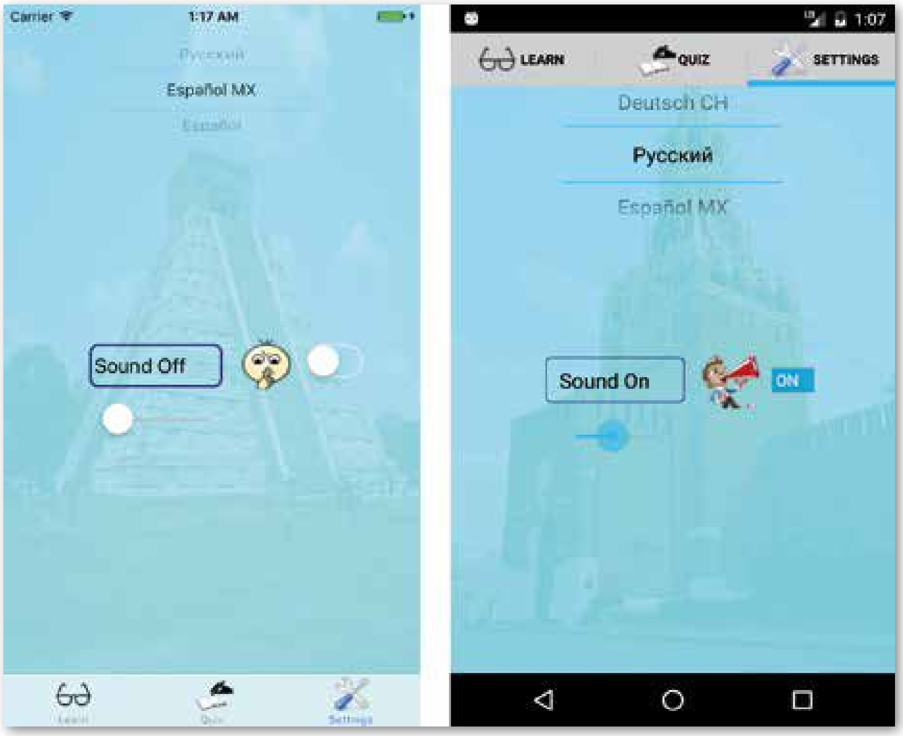


Figure 5: Various Widgets on iOS and Android

Listing 10: The implementation of InvokeNativeFunction class

```
public class InvokeNativeFunction : ParserFunction
{
    protected override Variable Evaluate(ParsingScript script)
    {
        string methodName = Utils.GetItem(script).AsString();
        Utils.CheckNotEmpty(script, methodName, m_name);

        string paramName = Utils.GetToken(script,
                           Constants.NEXT_ARG_ARRAY);
        Utils.CheckNotEmpty(script, paramName, m_name);

        Variable paramValueVar = Utils.GetItem(script);
        string paramValue = paramValueVar.AsString();

        var result = Utils.InvokeCall(typeof(Statics),
                           methodName, paramName, paramValue);
        return result;
    }
}
```

Calling the Native C# Functions from the CSCS Code

It may be more convenient to use already-existing C# code from the CSCS code and get the results back into CSCS. Even though any feature of C# can be implemented in CSCS, this may require some time, and the C# code may be already available.

Let's see an example of how the C# code receives an argument from CSCS, gets the current time, and returns a string back to CSCS. Then this string is shown as a button title. Here's the CSCS implementation:

```
clicks = 0;
function click(sender, arg) {
  clicks++;
  title = CallNative("ProcessClick", "arg",
                     clicks);
  SetText(sender, title);
}
loc1 = GetLocation("ROOT", "LEFT",
                   "ROOT", "BOTTOM");
AddButton(loc1, "but1", "Left", 220, 80);
AddAction(but1, "click");
```

```
loc2 = GetLocation("but1", "RIGHT",
                   "ROOT", "BOTTOM");
AddButton(loc2, "but2", "Right", 220, 80);
AddAction(but2, "click");
```

The result of executing this script and clicking a few times on each button is shown in **Figure 4**.

Let's see how it's implemented. **Listing 9** contains the C# implementation of invoking a method with one string parameter that returns a string. The implementation for methods with a different number of parameters or with different types of arguments, is analogous.

You can see that you cache the compiled function to be executed. The effect of caching is quite noticeable visually: The first click takes a longer time than the consequent ones, especially on Android. You can also pre-cache commonly called functions at the start-up phase.

This **InvokeCall()** method is called from a Parser function, which is the first point of contact in the C# code with CSCS. The iOS and Android implementation is the same and it's shown in **Listing 10**.

To glue everything together, you need to register the InvokeNativeFunction with the parser:

```
ParserFunction.RegisterFunction("CallNative",
               new InvokeNativeFunction());
```

What's left is the actual implementation of the method being called from the CSCS code. As you saw in **Listing 10**, the method is implemented in the Statics class that I added in the shared project area. In the Statics class, you can implement all of the methods called from C# and the same code is called from both iOS and Android:

```
public class Statics
{
  public static string ProcessClick(string arg)
  {
    var now = DateTime.Now.ToString("T");
    return "Clicks: " + arg + "\n" + now;
  }
}
```

Very similarly, you can also implement calling functions with a different number of arguments or with different argument types.

Where the CSCS Script Execution is Triggered in C# Code

One important question is: Where exactly in the flow do you execute the CSCS script?

For iOS, the answer is easier than for Androi.: In iOS, it can be done just at the end of the AppDelegate.FinishedLaunching() method.

For Android, the first attempt to run the CSCS script at the end of the MainActivity.OnCreate() method failed. The reason was that the global layout has not been completely initialized in the OnCreate() method.

References

How to Write Your Own Programming Language in C#: http://www.codemag.com/article/1607081

Microsoft Visual Studio Community 2017 License Terms: https://www.visualstudio.com/license-terms/mlt553321/

Android Layout: https://developer.android.com/guide/topics/ui/declaring-layout.html

Apple Auto Layout: https://developer.apple.com/library/content/documentation/UserExperience/Conceptual/AutolayoutPG/index.html

GitHub CSCS Source Code: https://github.com/vassilych/mobile

Installing Visual Studio 2017 with Xamarin: https://blog.xamarin.com/installing-visual-studio-2017-made-easy

Listing 11: Playing with Widgets: CSCS code

```
sound_clicks = 0;
function sound_click(sender, arg) {
  sound_clicks++;
  if (sound_clicks % 2 == 0) {
    enable_sound();
  } else {
    disable_sound();
  }
}
function enable_sound() {
    SetText(buttonCenterLeft, "Sound On");
    SetImage(imgView, "sound_on");
    SetValue(switch, 1);
}
function disable_sound() {
    SetText(buttonCenterLeft, "Sound Off");
    SetImage(imgView, "sound_off");
    SetValue(switch, 0);
}
function slider_change(sender, arg) {
   if (GetValue(slider) > 1) {
      enable_sound();
   } else {
      disable_sound();
   }
}
function pickerMove(row) {
  SetBackground(countryImages[row]);
}

countryImages = {"us_bg", "gb_bg", "de_bg", "ch_bg", "ru_bg", "mx_bg", "es_
bg", "br_bg", "fr_bg", "it_bg", "cn_bg", "jp_bg", "ar_bg"};
countries = {"English US", "English", "Deutsch", "Deutsch CH", "Русский",
"Español MX", "Español", "Português BR", "Français", "Italiano", "中文", "日
本語", "العربية"};

locPicker = GetLocation("ROOT", "CENTER", "ROOT", "TOP", 0, -20);
AddWidget("TypePicker", locPicker, "pickerColor", "Picker",
          380, 280);
AddWidgetData(pickerColor, countries, "pickerMove");
pickerMove(0);

locCenterLeft = GetLocation("ROOT", "CENTER", "ROOT", "CENTER",
                            -100, 0);
AddButton(locCenterLeft, "buttonCenterLeft", "Sound On", 240, 80);
AddBorder(buttonCenterLeft, 2, 8, "#000080");
AlignText(buttonCenterLeft, "left");
AddAction(buttonCenterLeft, "sound_click");

locCenterRight = GetLocation("ROOT", "CENTER", "ROOT", "CENTER",
                             100, 0);
AddImageView(locCenterRight, "imgView", "sound_on", 100, 100);

locCenter2 = GetLocation(imgView, "RIGHT", imgView, "CENTER",
                         20, 0);
AddSwitch(locCenter2, "switch", "1", 80, 80);
AddAction(switch, "sound_click");

locCenterDown = GetLocation(buttonCenterLeft, "CENTER", buttonCenterLeft,
"BOTTOM", 0, 20);
AddSlider(locCenterDown, "slider", "0:100", 200, 80);
AddAction(slider, "slider_change");
```

CSCS Function	Description
GetLocation(referenceX, relationX, referenceY, relationY, marginX, marginY, parentView)	Creates a location relative to the other widget "referenceX" locations horizontally and "referenceY" vertically. Optionally, additional margins on the X and Y axes can be supplied. If the parentView is specified, the widget is constructed inside of it.
AddWidget(type, widgetName, location, initString, width, height)	A generic function to add a widget at a given location with the initialization string and with a given width and height. Particular specializations for different widget types follow.
AddView(widgetName, location, initString, width, height)	Adds a UIView on iOS and a RelativeLayout on Android to place additional widgets inside of it.
AddButton(widgetName, location, initString, width, height)	Adds a UIButton on iOS and a Button on Android with a given title string from the initString. The button has a border (it can be removed with AddBorder function below).
AddLabel(widgetName, location, initString, width, height)	Adds a UILabel on iOS and a TextView on Android with a given text string from the initString.
AddTextView(widgetName, location, initString, width, height)	Adds a UITextView on iOS and a TextEdit on Android with a given text string from the initString.
AddTextEdit(widgetName, location, initString, width, height)	Adds a UITextField on iOS and a TextEdit on Android with a given text hint (or a placeholder in iOS terms) from the initString.
AddImagetView(widgetName, location, initString, width, height)	Adds a UIImageView on iOS and an ImageView on Android with a given image filename from the initString.
AddSwitch(widgetName, location, initString, width, height)	Adds a UISwitch on iOS and a Switch on Android. The initString argument must be either a 0 or a 1, indicating the initial switch state.
AddSlider(widgetName, location, initString, width, height)	Adds a UISlider on iOS and a SeekBar on Android. The initString argument must have format "fromNumber:toNumber", e.g. "0:100".
AddStepper(widgetName, location, initString, width, height)	Adds a UIStepper on iOS and two Buttons on Android. The initString argument must have format "fromNumber:toNumber", e.g. "0:100".
AddStepperLeft(widgetName, location, initString, width, height)	Adds a stepper as in AddStepper but also with the label with the stepper current value on the left of the stepper.
AddStepperRight(widgetName, location, initString, width, height)	Adds a stepper as in AddStepper but also with the label with the stepper current value on the right of the stepper.
AddCombobox(widgetName, location, initString, width, height)	Adds a UITypePicker on iOS and a Spinner on Android. It can be populated via AddWidgetData and AddWidgetImages functions.
AddSegmentedControl(widgetName, location, initString, width, height)	Adds a UISegmentedControl on iOS and either a Switch (for two segments) or buttons on Android. InitString specifies number of segments.
AddTypePicker(widgetName, location, initString, width, height)	Adds a UITypePicker on iOS and a NumberPicker on Android. The initString argument is not used. Used together with AddWidgetData.
AddListView (widgetName, location, initString, width, height)	Adds a UITableView on iOS and a ListView on Android. It can be populated via AddWidgetData and AddWidgetImages functions.

Table 1: CSCS Cross-Platform Functions for Adding Widgets

The trick is to register a listener that will be triggered as soon as the global layout is initialized:

```
protected override void OnCreate(Bundle
                        savedInstanceState)
{
  base.OnCreate(savedInstanceState);
  //some other stuff

  ViewTreeObserver ob =
           relativelayout.ViewTreeObserver;
  ob.AddOnGlobalLayoutListener(
           new LayoutListener());
}
```

In the listener code, first you unregister the listener (otherwise it will be triggered on every change to the layout) and then run the CSCS script:

```
public class LayoutListener : Java.Lang.Object,
       ViewTreeObserver.IOnGlobalLayoutListener
{
  public void OnGlobalLayout()
  {
    var ob =
        MainActivity.TheLayout.ViewTreeObserver;
    ob.RemoveOnGlobalLayoutListener(this);
    MainActivity.RunScript();
  }
}
```

In the RunScript method, you register all of the Parser functions right before the parser execution is started. Here's a fragment from the RunScript method:

```
public static void RunScript()
{
  ParserFunction.RegisterFunction("_IOS_", new
     CheckOSFunction(CheckOSFunction.OS.IOS));

  // ... Registration of all other functions
  // with the Parser here ...

  string script = "";
  AssetManager assets = TheView.Assets;
  using (StreamReader sr = new StreamReader(
              assets.Open("script.cscs"))) {
    script = sr.ReadToEnd();
  }
  Interpreter.Instance.Process(script);
}
```

Widgets, Widgets, Widgets

Once you know how to build the layout, another important part of the CSCS scripting language is to be able to use as many widgets as possible. You can see what's currently implemented in **Table 1, 2,** and **3**, but I'm sure that by the time you read this article, many more widgets will be implemented. Just check the accompanying source-code download at the GitHub link or go to the CODE Magazine website and download it from the article's link.

As a final example, let's see how the widgets shown in **Figure 5** are implemented.

There's a TypePicker on the top: As soon as you change the value there by moving the picker's wheel, a different background image is shown.

Also, there is a Button, an ImageView, a Switcher, and a Slider. As soon as you click the Button, or the Switcher, or change the value of the Slider, the other widgets also change.

Listing 11 shows how it's implemented in CSCS.

Finally, **Tables 1, 2**, and **3** contain all currently available functions for mobile development in CSCS.

I'm sure by the time you read this, there'll be many more functions available: Don't forget to check out my github.com page in the links section.

Wrapping Up

Using the CSCS scripting language, you can write cross-platform applications that run natively on any device.

You can proceed as follows: Download the sample project in the accompanying source code download section and start playing with the script.cscs file there.

CSCS Function	Description
AddWidgetData(widgetName, data, selectAction)	Adds a list of string data to the widget. Also registers the selectAction CSCS function to be called when an entry is selected.
AddWidgetImages(widgetName, images)	Adds a list of images (specified as image filenames) to the widget.
AddAction(widgetName, callback)	Adds a widget action on clicking event. If the widgetName is ROOT, then adds a global action, e.g., for a Tab app is OnTabSelected.
AutoScale(scale)	Automatically scales all the widgets according to the display resolution. The scale parameter is optional and it's 1.0 by default. The widgets can be scaled individually as well.
AddLongClick(widgetName, callback)	Adds a widget action on a long clicking event, i.e., when the user presses and holds for a few seconds.
AddSwipe(widgetName, callback)	Adds detection of swiping left, right, up, and down. Calls callback, passing as an argument what type of the swipe event happened.
AddDragAndDrop(widgetName, callback)	Adds drag and drop functionality. Moves widget till the finger is lifted.
AddBorder(widgetName, width, corner, color)	Adds a border around a widget. If the width is 0, removes existing border. The color is optional and is black by default.
AddTab(tabName, activeImage, inactiveImage)	Adds a tab to the app with corresponding active and inactive images.
GetSelectedTab()	Returns index of a tab that is active in the running app.
SelectTab(index)	Programmatically activates a tab.
AlignText(widgetName, alignType)	Aligns text according to the alignType, which can be left, right, center, justified, fill or natural.
GetText(widgetName)	Returns widget's text.
SetText(widgetName, text)	Sets text to the widget.
GetValue(widgetName)	Returns widget value (an integer, Boolean, or a double).
SetValue(widgetName, value)	Sets value to the widget (an integer, Boolean, or a double).
SetImage(widgetName, imageFile)	Sets image on a widget.
SetBackground(imageFile)	Sets background image on the root view.
SetBackgroundColor(widgetName, color)	Sets background color on a widget. If the widget name is ROOT, sets background color on the root view.
SetSize(widgetName, width, height)	Sets widget's height and width.
SetFontSize(widgetName, fontSize)	Sets the font size of a widget.
ShowView(widgetName)	Shows a widget (or a view / layout).
HideView(widgetName)	Hides a widget (or a view / layout).
Move(widgetName, x, y)	Moves a widget x pixels right and y pixels down (for a negative, x moves it left and for a negative y, moves it up).
RemoveView(widgetName)	Removes passed view (or widget name) from the layout.
RemoveAllViews()	Removes all views from the layout. This can be used on the orientation change, when rebuilding the layout.

Table 2: CSCS Cross-Platform Functions: UI Manipulation

CSCS Function	Description
CallNative(methodName, argName, argValue)	Calls a native C# method, having given parameters. The C# method has one string argument and returns a string.
GetRandom(limit,numberOfValues)	A pseudo-random number generator returning a list of generated numbers between 0 and limit (exclusive limit).
ShowToast(message, duration, bg_color, fg_color)	Shows a Toast on Android and a custom Toast implementation on iOS. The last two parameters are optional.
AlertDialog(type, title, message, button1, action1, button2, action2)	Shows an alert dialog to the user. The last three parameters are optional; the dialog is dismissed if there is no action.
Speak(phrase, voice, rate, pitch)	Adds Text-To-Speech functionality. Only phrase is a mandatory parameter. The default voice is “en-US”. The speech rate and pitch are between 0 and 1.
VoiceRecognition(callback, voice)	Starts voice recognition and calls the callback function on completion, passing the recognized phrase as an argument. The default voice is “en-US”.
ReadFile(filename)	Reads file form the device assets directory. Returns a list of lines of that file.
Schedule(timeout, callback, timerId)	Schedule execution of the callback function after the timeout milliseconds on the main GUI thread. The timerId is passed as an argument to the callback.
GetSetting(settingName, type, defaultValue)	Gets setting value from the device settings settings. Type can be “float”, “int”, “string” or “bool”. The defaultValue is optional.
SetSetting(settingName, value, type)	Saves passed setting on the device so that later it can be retrieved by the GetSetting() function.
GetDeviceLocale()	Returns the language locale of the device.
SetAppLocale(localeName)	Sets the locale of the app. Used closely with the Localized() function. The expected format of the localeName is “en-US”, “es-MX”, etc.
Localize(text, language)	Localizes passed text to the current program language (by default device language). If an optional parameter “language” is set, localizes to that language.
InitIAP(publicKey)	Initializes In-App Purchase (Billing) service, connecting to the Apple App Store or to the Google Play Store.
Restore(callback , productId1, productId2, ...)	Checks if given products have been purchased before, connecting to the Apple App Store or to the Google Play Store.
Purchase(callback, productId)	Purchases a product with the given product ID, connecting to the Apple App Store or to the Google Play Store.
InitAds(appId, interstitialId, bannerId)	Inits the Google AdMob advertisement framework with the initialization parameters, previously requested at https://www.google.com/admob
AddBanner(widgetName, location, bannerType)	Adds a BannerView on iOS and an AdView on Android for the Google AdMob network. The bannerType can be either SmartBanner, MediumRectangle, Banner, LargeBanner, FullBannner, or Leadeboard.
ShowInterstitial()	Shows an interstitial (full screen) advertisement from the Google AdMob network.
OnOrientationChange(callback)	Calls the callback function when there’s a widget orientation change, passing as a parameter the new orientation (portrait or landscape).
Orientation	Current device orientation (e.g. Landscape or Portrait).
DisplayWidth	Returns the width of the display in pixels.
DisplayHeight	Returns the height of the display in pixels.
ANDROID	Returns true if and only if the current code is being executed on an Android device.
IOS	Returns true if and only if the current code is being executed on iOS.
VERSION	Returns version of the smartphone operating system.

Table 3: CSCS Cross-Platform not GUI Modification Related Functions

The features presented in this article constitute a small fraction of what you can do with CSCS. I plan to expose more advanced topics in the next articles. Some of these future topics are: advanced controls and widgets, in-app purchases, text-to-speech, voice recognition, localization, easy ways of having different layouts in different device orientations, and adding advertising content (like Google AdMob), just to name a few. Everything can be made in CSCS.

But more importantly, it’s relatively straightforward to modify the existing functionality of CSCS, or to add new functions.

To add a new function to CSCS, you need to create a class deriving from the ParserFunction class and override its Evaluate method for all of the platforms where you want your script run. Then, you register the new function with the parser, as you’ve seen in this article.

Another advantage of CSCS is that not all of the code must be written in it, but it can be combined with other code written in C#. For example, you can use CSCS for the pure GUI development. You also saw how you can call a C# function from the CSCS code and virtually eliminate any overhead due to the marshalling by pre-compiling the Reflection functions.

I’m looking forward to getting any feedback you have for programming in CSCS for the mobile development.

Vassili Kaplan

Print impresses.

Advertise in CODE Magazine and see how print can **wow** your customers.

Contact tammy@codemag.com for advertising opportunities.

Implementing Machine Learning Using Python and Scikit-learn

In my previous article ("Getting Started with Machine Learning Using Microsoft Azure ML Studio", http://www.codemag.com/article/1709071), I explained the concepts behind machine learning and got you started using the Microsoft Azure Machine Learning Studio (MAML). For beginners, the MAML is a really good way to dabble with machine learning without

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.

possessing the mathematical prerequisites that are usually required of a data scientist. However, to really implement machine learning, you need to move beyond MAML and be able to implement your learning models programmatically. This has the advantage of fine-tuning the models to your needs, and, at the same time, affording you the flexibility to deploy the models in whatever manner you want.

One of the languages that's most popular with data scientists is Python. With its vast amount of third-party library support, Python is well-suited for implementing machine learning. In this article, I'll build a couple of models using Python and its accompanying library Scikit-learn. Although Python is popular among data scientists, another language remains popular among statisticians: R. I don't have the luxury of space to delve into R programming in this article, but I'll provide the solution for the Titanic problem in both Python and R so that enthusiasts can devour them over the weekends.

Introduction to Scikit-learn

In one of my earlier articles on Data Science ("Introduction to Data Science using Python", http://www.codemag.com/article/1611081), you learned how to use Python together with libraries such as NumPy and Pandas to perform number crunching, data visualization, and analysis. For machine learning, you can also use these libraries to build learning models. However, doing so requires that you have a strong appreciation of the mathematical foundation for the various machine learning algorithms. This isn't a trivial matter. Instead of implementing the various machine-learning algorithms manually, fortunately, someone else has already done the hard work for you.

Scikit-learn is a Python library that implements the various types of machine learning algorithms, such as classification, regression, clustering, decision tree, and more. Using Scikit-learn, implementing machine learning is now simply a matter of supplying the appropriate data to a function so that you can fit and train the model.

Getting Datasets

Often, one of the challenges in machine learning is obtaining sample datasets for experimentation. Fortunately, Scikit-learn comes with a few standard sample datasets, which makes learning machine learning easy.

To load the sample datasets, import the **datasets** module and load the desired dataset. For example, the following code snippets load the Iris dataset:

```
from sklearn import datasets
iris = datasets.load_iris()
print(iris)         # raw data of type Bunch
```

The dataset loaded is a Bunch object, which is a Python dictionary that provides attribute-style access. You can use the **DESCR** property to obtain a description of the dataset:

```
print(iris.DESCR)
```

More importantly, you can obtain the features of the dataset using the **data** property:

```
print(iris.data)    # Features
```

The above statement prints the following:

```
[[ 5.1  3.5  1.4  0.2]
 [ 4.9  3.   1.4  0.2]
   ...
 [ 6.2  3.4  5.4  2.3]
 [ 5.9  3.   5.1  1.8]]
```

You can also use the **feature_names** property to print the names of the features:

```
print(iris.feature_names)      # Feature Names
```

The above statement prints the following:

```
 ['sepal length (cm)', 'sepal width (cm)',
 'petal length (cm)', 'petal width (cm)']
```

This means that the dataset contains four columns: **sepal length**, **sepal width**, **petal length**, and **petal width**. To print the label of the dataset, use the **target** property. For the label names, use the **target_names** property:

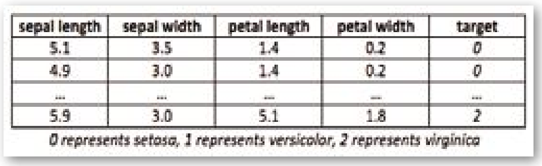
```
print(iris.target)             # Labels
print(iris.target_names)       # Label names
```

The above prints out the following:

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
2 2 2 2 2 2 2 2]
['setosa' 'versicolor' 'virginica']
```

Note that not all sample datasets support the **feature_names** and **target_names** properties.

Figure 1 summarizes how the dataset looks:



sepal length	sepal width	petal length	petal width	target
5.1	3.5	1.4	0.2	0
4.9	3.0	1.4	0.2	0
...	...	...	...	...
5.9	3.0	5.1	1.8	2

0 represents setosa, 1 represents versicolor, 2 represents virginica

Figure 1: The fields in the Iris dataset and its target

Often, it's useful to convert the data to a Pandas DataFrame so that you can manipulate it easily:

```
import pandas as pd
df = pd.DataFrame(iris.data)
             # convert features
             # to dataframe in Pandas
print(df.head())
```

The above statements print out the following:

```
     0    1    2    3
0  5.1  3.5  1.4  0.2
1  4.9  3.0  1.4  0.2
2  4.7  3.2  1.3  0.2
3  4.6  3.1  1.5  0.2
4  5.0  3.6  1.4  0.2
```

Besides the Iris dataset, you can also load some interesting datasets, such as the following:

```
# data on breast cancer
breast_cancer = datasets.load_breast_cancer()

# data on diabetes
diabetes = datasets.load_diabetes()

# dataset of 1797 8x8 images of hand-written
# digits
digits = datasets.load_digits()
```

Solving Regression Problems Using Linear Regression

The easiest way to get started is with *Linear Regression*. Linear Regression is a linear approach for modeling the relationship between a scalar dependent variable Y and one or more explanatory variables (or independent variables). For example, let's say that you have a set of data consisting of the heights of a group of people and their corresponding weights:

```
%matplotlib inline
import matplotlib.pyplot as plt

# represents the heights of a group of people
heights = [[1.6], [1.65], [1.7], [1.73], [1.8]]

# represents the weights of a group of people
weights = [[60], [65], [72.3], [75], [80]]

plt.title('Weights plotted against heights')
plt.xlabel('Heights in metres')
plt.ylabel('Weights in kilograms')

plt.plot(heights, weights, 'k.')
```

```
# axis range for x and y
plt.axis([1.5, 1.85, 50, 90])
plt.grid(True)
```

When you plot a chart of weights against heights, you'll see the figure shown in **Figure 2**.

From the chart, you can see that there's a positive co-relation between the weights and heights for that group of people. You could draw a straight line through the points and use that to predict the weight of another person based on height.

Using the LinearRegression Class for Fitting the Model

So how do you draw the straight line that cuts though all the points? It turns out that the Scikit-learn library has a **LinearRegression** class that helps you to do just that. All you need to do is to create an instance of this class and use the **heights** and **weights** lists to create a linear regression model using the **fit()** function, like this:

```
from sklearn.linear_model
    import LinearRegression

# Create and fit the model
model = LinearRegression()
model.fit(heights, weights)
```

Once you've fitted the model, you can start to make predictions using the **predict()** function, like this:

```
# make prediction
weight = model.predict([[1.75]])[0][0]
print(weight)
# 76.0387650086
```

Plotting the Linear Regression Line

It would be useful to visualize the linear regression line that's been created by the LinearRegression class. Let's do this by first plotting the original data points and then sending the **heights** list to the model to predict the weights. Then plot the series of forecasted weights to obtain the line. The following statements show how this is done:

```
import matplotlib.pyplot as plt

heights = [[1.6], [1.65], [1.7], [1.73], [1.8]]
```

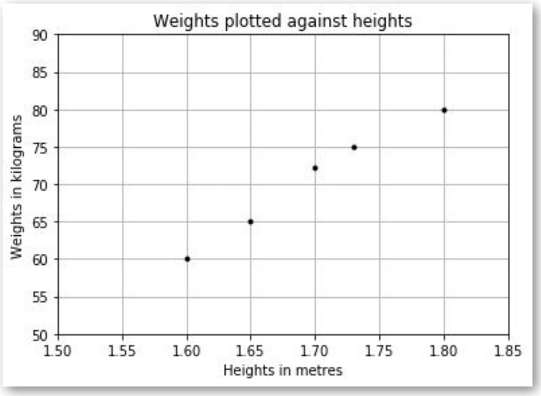


Figure 2: Plotting the weights against heights for a group of people

The Fisher's Iris Flower Dataset

The Fisher's (a British statistician and biologist) Iris flower data set consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica, and Iris versicolor). Four features were measured from each sample: the length and the width of the sepals and petals, in centimeters. Based on the combination of these four features, Fisher developed a linear discriminant model to distinguish the species from each other.

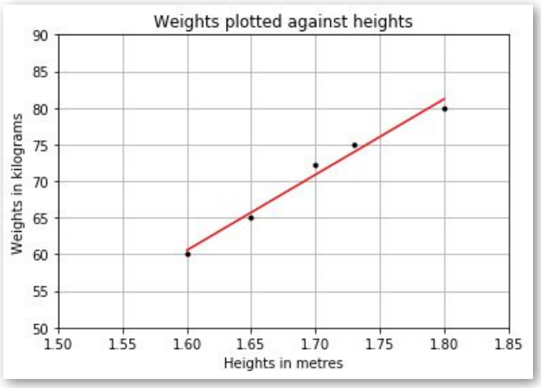


Figure 3: Plotting the linear regression line

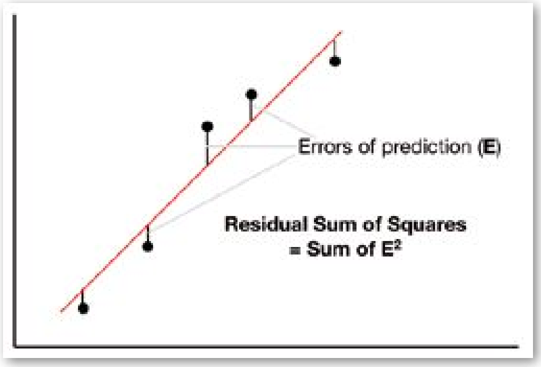


Figure 4: Calculating the Residual Sum of Squares for Linear Regression

```
weights = [[60], [65], [72.3], [75], [80]]

plt.title('Weights plotted against heights')
plt.xlabel('Heights in metres')
plt.ylabel('Weights in kilograms')

plt.plot(heights, weights, 'k.')

plt.axis([1.5, 1.85, 50, 90])
plt.grid(True)

# plot the regression line
plt.plot(heights, model.predict(heights),
         color='r')
```

Figure 3 shows the linear regression line.

Examining the Performance of the Model by Calculating the Residual Sum of Squares

To know whether your linear regression line is well fitted to all the data points, use the **Residual Sum of Squares (RSS)** method. **Figure 4** shows how the RSS is calculated.

The following code snippet shows how the RSS is calculated in Python:

```
import numpy as np

print('Residual sum of squares: %.2f' %
       np.sum((weights - model.predict(heights))
       ** 2))
# Residual sum of squares: 5.34
```

$$R^2 \equiv 1 - \frac{RSS}{\text{TSS}}$$

$$\text{TSS} = \sum_{i=1}^{n}(y_i - \bar{y})^2$$

$$RSS = \sum_{i=1}^{n}(y_i - f(x_i))^2$$

Figure 5: The formula for calculating R-Squared

The RSS should be as small as possible, with 0 indicating that the regression line fits the points exactly (rarely achievable in the real world).

Evaluating the Model Using a Test Dataset

Now that the model is fitted with the training data, you can put it to the test. You need the following test dataset:

```
# test data
heights_test = [[1.58], [1.62], [1.69], [1.76],
                [1.82]]
weights_test = [[58], [63], [72], [73], [85]]
```

You can measure how closely the test data fits the regression line using the **R-Squared** method. The **R-Squared** method is also known as the *coefficient of determination* or the *coefficient of multiple determinations* for multiple regressions.

The formula for calculating R-Squared is shown in **Figure 5**.

Using the formula shown for R-Squared, you can calculate R-Squared in Python using the following code snippet:

```
# total sum of squares
weights_test_mean =
    np.mean(np.ravel(weights_test))
ss_total = np.sum((np.ravel(weights_test) -
                   weights_test_mean) ** 2)
print("ss_total: %.2f" % ss_total)

# total sum of residuals
ss_res = np.sum((np.ravel(weights_test) -
                 np.ravel(model.predict(
                     heights_test))) ** 2)
print("ss_res: %.2f" % ss_res)

# r_squared
r_squared = 1 - (ss_res / ss_total)
print("R-squared: %.2f" % r_squared)
```

The explanation for deriving the formula for R-Squared is beyond the scope of this article, but you can visit https://en.wikipedia.org/wiki/Coefficient_of_determination for more details.

The previous code snippet yields the following result:

```
ss_total: 430.80
ss_res: 24.62
R-squared: 0.94
```

Fortunately, Scikit-learn has the **score()** function to calculate the R-Squared automatically for you:

```
# using scikit-learn to calculate r-squared
print('R-squared: %.4f' %
    model.score(heights_test,
                weights_test))

# R-squared: 0.9429
```

A R-Squared value of 94% indicates a pretty good fit for your test data.

Solving Classification Problems Using Logistic Regression

In the previous section, you saw how linear regression works in Scikit-learn. Starting with linear regression is a good way to understand how machine learning works in Python. In this section, you're going to solve the Titanic prediction problem using another machine learning algorithm: Logistic Regression. This is the same problem that I solved in my previous article using the Microsoft Azure Machine Learning Studio (MAML). Logistic Regression is a type of classification algorithm that involves predicting the outcome of an event, such as whether a passenger will survive the Titanic disaster or not.

If you need a refresher on the Titanic problem, be sure to check out my article on machine learning in the Sep/Oct 2017 issue of CODE Magazine.

Getting the Titanic Dataset

You can get the Titanic dataset by going to https://www.kaggle.com/c/titanic/data. Once you've obtained it, you can load it into a Pandas DataFrame:

```
import pandas as pd
from sklearn import linear_model
from sklearn import preprocessing

# read the data
df = pd.read_csv("titanic_train.csv")
print(df.head())
```

It's my habit to print out the data frame at every point of the process to verify that the data is properly loaded (or cleaned, as you'll see in the next couple of sections).

Cleaning the Data

Once the data is loaded, it's time to clean the data. Among all the different fields in the Titanic dataset, there are a number of columns that aren't important in building the machine learning model. For this purpose, let's drop the columns using the following code snippets:

```
# drop the columns that are not useful to us
df = df.drop('PassengerId', axis=1)
# axis=1 means column

df = df.drop('Name',        axis=1)
df = df.drop('Ticket',      axis=1)
df = df.drop('Cabin',       axis=1)

# check to see if the columnns are removed
print(df.head())
```

You should now see the dataset shown in **Listing 1**.

Because the dataset also contains rows with missing data, let's drop them and re-index the data frame:

```
df = df.dropna()              # drop all rows
                              # with NaN
df = df.reset_index(drop=True) # re-index the
                              # dataframe
print(df.head(10))
```

Encoding the Non-Numeric Fields

In order to perform logistic regression in Python, Scikit-learn needs the features to be encoded in numeric values. Examining the dataset, you'll see that values of **Sex** and **Embarked** are string types, and need to be encoded before you can go any further. For this purpose, you can use the **LabelEncoder** class to perform the conversion, like this:

```
# initialize label encoder
label_encoder = preprocessing.LabelEncoder()

# convert Sex and Embarked features to numeric
sex_encoded =
    label_encoder.fit_transform(df["Sex"])
print(sex_encoded)
# 0 = female
# 1 = male
df['Sex'] = sex_encoded

embarked_encoded = label_encoder.fit_transform(df["Embarked"])
print(embarked_encoded)
# 0 = C
# 1 = Q
# 2 = S
df['Embarked'] = embarked_encoded

print(df.head())
```

The last statement in the above code snippet prints out the output shown in **Listing 2**.

Note that the values for the **Sex** and **Embarked** fields are now replaced with the encoded values.

Listing 1: The output of the dataset after dropping some columns

```
   Survived  Pclass     Sex   Age  SibSp  Parch     Fare Embarked
0         0       3    male  22.0      1      0   7.2500        S
1         1       1  female  38.0      1      0  71.2833        C
2         1       3  female  26.0      0      0   7.9250        S
3         1       1  female  35.0      1      0  53.1000        S
4         0       3    male  35.0      0      0   8.0500        S
```

Listing 2: The values for the Sex and Embarked fields are now replaced with the encoded values

```
   Survived  Pclass  Sex   Age  SibSp  Parch     Fare  Embarked
0         0       3    1  22.0      1      0   7.2500         2
1         1       1    0  38.0      1      0  71.2833         0
2         1       3    0  26.0      0      0   7.9250         2
3         1       1    0  35.0      1      0  53.1000         2
4         0       3    1  35.0      0      0   8.0500         2
```

Making Fields Categorical

The next types of values that you need to take care of in the dataset is that of *categorical values*. Categorical types can only take on a limited, fixed number of possible values. Categorical values indicate to Scikit-learn that for this type of fields, numerical operations are not possible. A good example of a categorical field is Survived, where the value can only be 0 or 1 (and not anywhere in-between).

To make a field categorical, use the **Categorical** class in Pandas:

```
# make fields categorical
df["Pclass"]   = pd.Categorical(df["Pclass"])
df["Sex"]      = pd.Categorical(df["Sex"])
df["Embarked"] = pd.Categorical(df["Embarked"])
df["Survived"] = pd.Categorical(df["Survived"])

print(df.dtypes)       # examine the datatypes
                       # for each feature
```

The above prints out the data types for each field, confirming that the specified four fields are converted to category:

```
Survived    category
Pclass      category
Sex         category
Age          float64
SibSp          int64
Parch          int64
Fare         float64
Embarked    category
dtype: object
```

Splitting the Dataset into Train and Test Sets

With the dataset cleaned, you're now ready to split the dataset into two distinct sets: one for training and one for testing. But before that, you need to separate the dataset into two data frames: one containing all the features and one for the label:

```
# we use all columns except Survived as
# features for training
features = df.drop('Survived',1)

# the label is Survived
label    = df['Survived']
```

In Python, to split the rows for training and testing, you can use the **train_test_split()** function from the **model_selection** module:

```
from sklearn.model_selection
    import train_test_split

# split the dataset into train and test sets
train_features,test_features,
train_label,test_label =
    train_test_split(
        features,
        label,
        test_size = 0.25, # split ratio
        random_state = 1, # Set random seed
        stratify = df["Survived"])
```

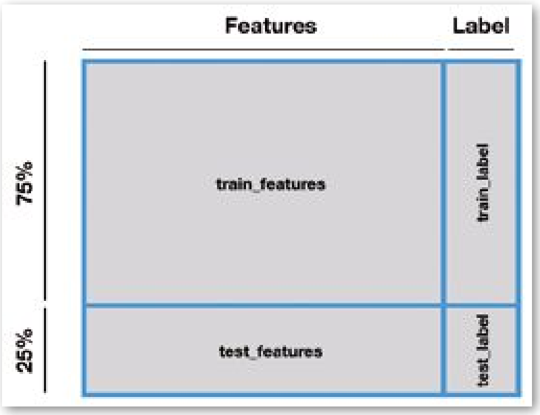


Figure 6: Splitting the dataset into training and testing sets

```
# Training set
print(train_features.head())
print(train_label)
```

The **stratify** argument lets you specify a target variable to spread evenly across the train and test splits.

Figure 6 summarizes how the dataset has been split.

You can now examine the training set and its associated label:

```
    Pclass Sex   Age  SibSp  Parch     Fare
514      3   1  21.0      0      0   8.4333
382      3   1   9.0      5      2  46.9000
285      1   0  22.0      0      1  55.0000
142      2   1  30.0      0      0  13.0000
671      3   1  34.5      0      0   6.4375

Embarked
2
2
2
2
0

[0, 0, 1, 0, 0, ..., 0, 1, 1, 0, 0]
Length: 534
Categories (2, int64): [0, 1]
```

Likewise, examine the test set:

```
# Test set for validation
print(test_features.head())
print(test_label)
```

Observe that the training set has 534 rows and the test set has 178 rows, which is split in the ratio of 3:1:

```
    Pclass Sex   Age  SibSp  Parch     Fare
227      3   1  19.0      0      0   8.0500
318      2   1  46.0      0      0  26.0000
538      3   1  20.0      0      0   9.2250
199      1   1  37.0      1      1  52.5542
235      2   1  36.0      0      0  12.8750

Embarked
```

```
2
2
2
2
0

[1, 0, 0, 1, 0, ..., 0, 0, 1, 1, 0]
Length: 178
Categories (2, int64): [0, 1]
```

Training the Model

You can now go ahead and train the model. For this, use the **LogisticRegression** class. Train the model using the **train_features** against the **train_label**:

```
# initialize logistic regression model
log_regress = linear_model.LogisticRegression()

# Train the model
log_regress.fit(X = train_features ,
                y = train_label)
```

Once the model is fitted with the data (trained), you can check its intercept and coefficients:

```
# Check trained model intercept
print(log_regress.intercept_)

# Check trained model coefficients
print(log_regress.coef_)
```

The output should look something like this:

```
[ 3.87427541]
[[-0.85532931 -2.30146604 -0.03444764
  -0.29622236 -0.00644779  0.00482113
  -0.01987031]]
```

Making Predictions

With the model trained, you can make predictions using the test set. You use the **predict()** function and pass in the **test_features** set:

```
# Make predictions
preds = log_regress.predict(X=test_features)
print(preds)
```

The predictions are in the form of 0s (did not survive) and 1s (for survived):

```
[0 0 0 ... 1 0 1]
```

It's useful (as you'll see later when you plot the ROC curve) that you also get the probabilities of the prediction. To do that, use the **predict_proba()** function:

```
# Predict the probablities
pred_probs =
    log_regress.predict_proba(X=test_features)
print(pred_probs)
```

The result is in the form of [*Death Probability*, *Survival Probability*]:

```
[[ 0.83870368  0.16129632]
 [ 0.83710341  0.16289659]
 [ 0.84255956  0.15744044]
           ...
 [ 0.34218839  0.65781161]
 [ 0.72572388  0.27427612]
 [ 0.39219784  0.60780216]]
```

Displaying the Metrics

Using the predictions and the **test_label**, you can generate a confusion matrix using the **crosstab()** function:

```
# Generate table of predictions vs actual
print(pd.crosstab(preds, test_label))
```

The confusion matrix looks like this:

```
col_0   0   1
row_0
0      92  24
1      14  48
```

The accuracy of the prediction is (92+48) / (92+48+14+24) = 0.7865168. The **LogisticRegression** object also comes with the **score()** function to return the accuracy of the predictions:

```
# get the accuracy of the prediction
log_regress.score(X = test_features ,
                  y = test_label)
# 0.7865168539325843
```

The result above matches the result that you calculated manually.

Apart from using the **crosstab()** function to generate the confusion matrix, you can use the **confusion_matrix()** function from the **metrics** module in Scikit-learn:

```
from sklearn import metrics

# view the confusion matrix
metrics.confusion_matrix(
    y_true = test_label,    # True labels
    y_pred = preds)         # Predicted labels
```

The confusion matrix is contained within an array:

```
array([[92, 14],
       [24, 48]])
```

The metrics module also allows you to generate the other metrics such as **precision**, **recall**, and **f1-score**:

```
# View summary of common classification metrics
print(metrics.classification_report(
      y_true = test_label,
      y_pred = preds))
```

The output looks like this:

```
         precision    recall  f1-score   support

      0       0.79      0.87      0.83       106
      1       0.77      0.67      0.72        72

avg / total
              0.79      0.79      0.78       178
```

Estimators

The sklearn.linear_model.LinearRegression class is an estimator. Estimators predict a value based on the observed data.

In Scikit-learn, all estimators implement the **fit()** and **predict()** methods.

Listing 3: Logistic Regression using R for the Titanic Dataset

```
# load the titanic training set
training.data.raw <- read.csv('Titanic_train.csv',
                              header=T,
                              na.strings=c(""))
print(head(training.data.raw))

#-----------------------------------------------------------------#

# print number of rows
print(c("Number of rows: ", nrow(training.data.raw)))

#-----------------------------------------------------------------#

# examine which are the columns with NA values
sapply(training.data.raw, function(x) sum(is.na(x)))

    # the sapply() function applies the function
    # passed as argument to each column of the dataframe

#-----------------------------------------------------------------#

# get the number of unique values for each column
sapply(training.data.raw, function(x) length(unique(x)))

#-----------------------------------------------------------------#

# we are only interested in the following features -
# Survived (2), Pclass(3), Sex(5), Age(6), SibSp(7),
# Parch(8), Fare(10), Embarked(12)
data <- subset(training.data.raw, select=c(2,3,5,6,7,8,10,12))
print(head(data))

#-----------------------------------------------------------------#

# omit the rows with empty values
data <- na.omit(data)

# OR, fill the empty values for age with the average age
# data$Age[is.na(data$Age)] <- mean(data$Age, na.rm=T)
# na.rm=T means NA values should be stripped before computation

print(data)

#-----------------------------------------------------------------#

# remove all rows with Embarked empty
data <- data[!is.na(data$Embarked),]

# check if there are NaN in data
sapply(data, function(x) sum(is.na(x)))

# check the length of unique values for each column
sapply(data, function(x) length(unique(x)))

#-----------------------------------------------------------------#

# check if Sex, Pclass, and Embarked are categorical values
is.factor(data$Sex)       # read.csv() by default will
                          # encode the string variables as factors
is.factor(data$Embarked)  # read.csv() by default will encode the
                          # string variables as factors
is.factor(data$Pclass)
is.factor(data$Survived)

#-----------------------------------------------------------------#

data$Pclass <- factor(data$Pclass)      # make Pclass a catagorical
                                        # value
data$Survived <- factor(data$Survived)  # make Survied a categorical
                                        # value

is.factor(data$Pclass)
is.factor(data$Survived)

print(head(data))

#-----------------------------------------------------------------#

# get the number of rows in data
n <- nrow(data)

data.shuffled <- data[sample(n), ]       # shuffle the data
print(head(data.shuffled))

train.indices <- 1:round(0.75 * n)        # indices for the first
                                          # 75% of the rows
train <- data.shuffled[train.indices,]   # get the first 75% of
                                          # the rows

train.indices <- (round(0.75 * n) + 1):n # indices for the
                                          # remaining 25% of the
                                          # rows
test <- data.shuffled[train.indices, ]   # get the remaining 25%
                                          # of the rows

print(nrow(train))
print(nrow(test))

#-----------------------------------------------------------------#

# create a glm() instance - Generalized Linear Models
model <- glm(Survived ~ .,                       # Survived is the
                                                 # label and
             family=binomial(link='logit'),      # features are the
                                                 # rest of
             data=train)                         # the data column
summary(model)

#-----------------------------------------------------------------#

# make predictions
pred.results <- predict(model,  # don't pass in the Survived (1)
                                # column
                        newdata=subset(
                            test,
                            select=c(2,3,4,5,6,7,8)),
                        type='response')

pred.results <- ifelse(pred.results > 0.5,1,0)  # if value>0.5
                                                # assign 1 else 0
print(pred.results)

#-----------------------------------------------------------------#

# find all the ones where prediction
# is correct and calculate the mean
accuracy <- mean(pred.results == test$Survived)
print(paste('Accuracy',accuracy))

#-----------------------------------------------------------------#

# installing the ROCR package
# run this only once
# install.packages('ROCR', repos='http://cran.us.r-project.org')
```

Listing 3: continued

```
library(ROCR)

# make predictions
p <- predict(model, newdata=subset(
            test,select=c(2,3,4,5,6,7,8)), type="response")
print(p)

# transform the input data into a standardized format
# needed by ROCR
pr <- prediction(p, test$Survived)

# perform all kinds of predictor evaluations
prf <- performance(pr, measure = "tpr", x.measure = "fpr")
plot(prf)

auc <- performance(pr, measure = "auc")
auc <- auc@y.values[[1]]
auc

#----------------------------------------------------------------#

# installing the caret package
# run this only once
install.packages('caret', dependencies = TRUE)
library(caret)

p <- ifelse(p > 0.5,1,0)

# print the confusion matrix
confusionMatrix(p, test$Survived)
```

Displaying the Receiver Operating Characteristic (ROC) Curve

Another metric that's very useful to determine whether your model is well fitted is the Receiver Operating Characteristic (ROC) curve. The **metrics** module has the **roc_curve()** function that helps you to generate a ROC curve, as well as the **auc()** function that calculates the area under the ROC curve.

The following code snippet plots the ROC curve and displays the AUC (Area Under Curve) (see **Figure 7**):

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# convert the probabilities from ndarray to
# dataframe
df_prob = pd.DataFrame(
    pred_probs,
    columns=['Death', 'Survived'])

fpr, tpr, thresholds = roc_curve(
    test_label, df_prob['Survived'])

# find the area under the curve (auc) for the
# ROC
roc_auc = auc(fpr, tpr)

plt.title(
    'Receiver Operating Characteristic Curve')
plt.plot(fpr, tpr, 'black',
         label='AUC = %0.2f'% roc_auc)

plt.legend(loc='lower right')
plt.plot([0,1],[0,1],'r--')
plt.xlim([-0.1,1.1])
plt.ylim([-0.1,1.1])

plt.ylabel('True Positive Rate (TPR)')
plt.xlabel('False Positive Rate (FPR)')
plt.show()

print(fpr) # Increasing false positive rates such
           # that element i is the false positive
           # rate of predictions with score >=
           # thresholds[i].
print(tpr) # Increasing true positive rates such
           # that element i is the true positive
           # rate of predictions with score >=
           # thresholds[i].

print(thresholds)
```

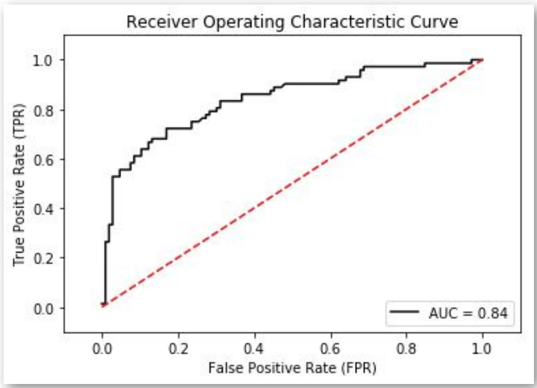


Figure 7: Plotting the ROC curve and displaying the AUC

For the R programmer, I've listed the solution for the Titanic problem written in R, as shown in **Listing 3**.

Summary

In this article, you've learned how to implement machine learning using Python and the Scikit-learn library. In particular, you've used the **LinearRegression** and **LogisticRegression** classes to solve regression and classification problems, respectively. In addition, the solution for the Titanic problem is also presented in R.

Wei-Meng Lee

R-Squared

R-squared is always between 0 and 100%: If it's 0%, that indicates that the model explains none of the variability of the response data around its mean. If it's 100%, that indicates that the model explains all the variability of the response data around its mean.

Software Archaeology

As developers, we're sometimes presented with the potentially unpleasant task of returning to really old code (or worse, as consultants, visiting it for the very first time). In this article, I take a look at a few projects and discuss some of the techniques used to get up and running quickly.

Chris G. Williams
chrisgwilliams@gmail.com
www.geekswithblogs.net/cwilliams
@chrisgwilliams

Chris G. Williams is a Senior Developer for Fluor Government Group, a nine-year multiple Microsoft MVP awardee (VB, XNA/DirectX, Windows Phone) and the author of *Professional Windows Phone Game Development*.

He has authored numerous articles on a variety of technologies, led a 14-city speaking tour, and has a series of mobile development webinars produced through DevExpress.com.

Chris is a regular presenter at conferences, code camps, and user groups around the country. He blogs at GeeksWithBlogs (.net) and also manages the MonoGame Indie Devs technical community on Facebook.

Excavating Old Code

I started writing code as a student/hobbyist in 1980, and professionally in 1990. For a lot of my career, the notion of working on old code was a relative one. If I work on something else for three months and then come back to it, that's old, right? I knew there were mainframe systems out there that were older than I was, but I hadn't seen any first hand.

My professional start centered on Visual Basic 3.0, and eventually HTML, JavaScript, and Classic ASP. Web technology was new, and changing so fast back then, that the concept of old code wasn't even on our radar.

In the early 2000s, Microsoft simultaneously announced the impending demise of "Classic" VB and heralded the arrival of VB.NET and C#. I jumped on that bandwagon, luckily dodging the "old code" bullet, because at that time, there was no legacy .NET code. Now, here we are and .NET is 15 years old. There's old code everywhere you look. In fact, .NET itself is now so bogged down with legacy junk that Microsoft is effectively starting over (again) with .NET Core.

In fact, .NET itself is now so bogged down with legacy junk that Microsoft is effectively starting over (again) with .NET Core.

One Piece at a Time

There's a great Johnny Cash song about a factory worker who builds Cadillac cars. He really wants one of his own, but can't afford it, so he smuggles out one part every week for years. When it finally comes time to assemble the car, unsurprisingly, none of the parts fit together quite right.

That car was this project I'm about to exhume. Sure, it worked—mostly—but maintenance was a nightmare.

This relatively small ASP.NET Web project had been around for a few years and had been touched by A LOT of different people. You could practically see each person's distinct fingerprints, in the form of coding style and architectural choices.

By the time I inherited it, there were at least four different ways of doing CRUD operations in the code. One guy was a big believer in Stored Procedures and another preferred inline SQL. Another person wanted everything in a data layer that he wrote himself, and another preferred using Enterprise Library.

Depending on your task, you may not have time to fix everything else you find that's horribly, horribly wrong, no matter how much you may want to.

```
' Case In Point:

If Not IsNothing(obj) = False Then
   ...

End If
```

If you're hunting down an issue, sometimes you have to break stuff (temporarily) in order to fix other stuff. If you aren't sure what something does, turn it off (comment it out) and see what breaks. If you royally screw something up and can't find your way back, there's always source control.

I'm making an assumption here, so if you don't have source control, stop what you're doing and get the project into source control NOW, or make a backup or whatever you have to do. If you're laughing, I'm assuming it's because you're already painfully aware of how often this happens in the real world.

If you're not there to fix a bug, it stands to reason that you must be trying to add something new. The existing architecture may make adding new things a painful process.

Depending on your task, you may not have time to fix everything else you find that's horribly, horribly wrong, no matter how much you may want to.

"Bolt-On Features" are often regarded as a bad, but sometimes they're an unavoidable necessity in this line of work. In a legacy application like this one (containing an architectural mish-mash), there's no single right approach aside from a total rewrite. Your best bet is to go with the style that most closely matches your own, or follow the architectural choices that you're most comfortable with. If you're lucky, this will be the newest tech, but don't count on it.

There are times when it's ok to show off your individuality and brilliance, but this probably isn't one of them. Adding more chaos to the pile could introduce unforeseen issues, make it harder for you to maintain later, and almost certainly make it harder for the next person to come along. As they say, "When in Rome, do as the Romans do."

Quality Software Consulting for Over 20 Years

CODE Consulting engages the world's leading experts to successfully complete your software goals. We are big enough to handle large projects, yet small enough for every project to be important. Consulting services include mentoring, solving technical challenges, and writing turn-key software systems, based on your needs. Utilizing proven processes and full transparency, we can work with your development team or autonomously to complete any software project.

Contact us today for your free 1-hour consultation.

Helping Companies Build Better Software Since 1993

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

CODE CONSULTING

Digging Deep

The next project I'm going to discuss is one of my own. I've been working on a small RPG game (called "Heroic Adventure!" or "HA!" for short) since 2003, off and on. I have periodic bursts of productivity, punctuated by years of inactivity.

Every time I revisit it after a long break, I realize just how much I've learned since the last time. Unfortunately, that means I start getting distracted by all the things I want to fix or refactor, instead of focusing on the reason I'm back in the code in the first place. Unless it's your intended goal, you must avoid the temptation to refactor just because you know a better way now.

A few months ago, I came back to the code after about two years away from it. There were some known bugs I wanted to fix, and some new features I wanted to add. Unfortunately, I hadn't written production VB.NET in quite some time, having been working mostly in C#, and the fact that this project was started in VB.NET 1.1, and upgraded to 2.0 (years ago) didn't help.

Unless it's your intended goal, you must avoid the temptation to refactor just because you know a better way now.

Half of the things I tried didn't exist in .NET 2.0, so I had to upgrade the project to 4.x. Fortunately, it didn't break anything (nor should it have), but that can and does happen.

I also spent a lot of time stepping through code in an effort to remember how it worked, or why I wrote it that way. A lot of people don't realize that F5 and Ctrl-F5 (debug and run without debugging, respectively) are not your only options for starting your code. You can also use F10 and F11 (step over and step into, respectively) which are especially handy if you aren't sure where exactly where in the codebase your application begins.

One thing I don't recommend is excessive use of the **<DebuggerStepThrough()>** attribute. It's OK for relatively short methods that you have to step through often, but there are few things more frustrating than stepping through code you barely remember (or have never seen) and bouncing off of one of these. You'll likely find yourself wanting to rip out every instance you find. If not, anyone who comes after you almost certainly will.

```
' Case In Point

' This is ok:
<DebuggerStepThrough()>
Function SimpleMethod() As Integer
   Return RND.Next(1, cap)
End Function

' This is not ok:
<DebuggerStepThrough()>
Function ComplexMethod() As Integer
   ' ...
   ' 380 lines of complex code
   ' ...
End Function
```

If you aren't familiar with it, this attribute essentially tells the debugger "everything is fine here, nothing to see, move along" and as a result, even if you're stepping into all of your methods line by line with F11, this behaves as though you hit F10 and simply executes the code without stepping into it.

You Want Me to Do What?

In this last example, I'd acquired responsibility for a collection of Microsoft Access 2010 forms applications. Like many Access apps, in addition to being really old code, they were initially written by a non-developer, passed through a few hands, and then into mine. To make matters worse, parts of the code were written in Albanian, which was definitely not listed on my resume. I also hadn't even installed Access on my developer computer in probably 10 years or more.

When I opened the first one, I had no idea what the app was supposed to do, how it worked, or what any of the **numerous** buttons did. So my first step was to insert breakpoints on every single method that served as an event handler and start clicking buttons so I could map everything out.

After I had a pretty good idea of what the buttons mapped to, the next step was to really dig into the meat of the code. In this example, I wasn't there to add new features to any of the apps, but simply to get them working in a new environment. It turns out, that while digging through the code, I discovered there was a lot of stuff that never worked in the first place (buttons that led to nowhere, lack of null-checking on various controls, incomplete methods, etc.).

I made a list of everything I found: environmental stuff (like hardcoded user IDs and network paths), broken stuff, unfinished or orphaned stuff, performance issues, etc. Once the list was done, the team prioritized what to address first, what to research further, and what to defer "indefinitely."

My first step was to insert breakpoints on every single method that served as an event handler and start clicking buttons so I could map everything out.

While exploring, I kept running into features that were restricted by user ID. Each of these features (there were many) had a hard-coded list of IDs in an If statement that wrapped all the code in the function. Instead of adding my ID to each list, which would have been time consuming and temporary, I put a breakpoint on each If statement and then dragged the execution point into the "True" branch, bypassing the security and allowing me to continue on my way.

```
' Case In Point

Private Sub cmdDoStuff_Click()
   If (ID = "uuuuu"
    Or ID = "vvvvv"
    Or ID = "wwwww"
    Or ID = "yyyyy"
    Or ID = "zzzzz") Then
      DoCmd.OpenForm "frmDoStuff", 0
   Else
      MsgBox "Access denied."
   End If
End Sub
```

Incidentally, Google Translate did a pretty decent job on the Albanian (except for the ones that were further subjected to Hungarian Notation), but it still took me a bit to wrap my head around some of the variable and function names.

Wrapping Up

Hopefully, you've found some useful tips in this article and you can make use of some or all of them the next time you have to dive into some old or unfamiliar code, or better yet, may you never get stuck working on someone else's old code. Good luck.

Chris G. Williams

Controlling the Flow of Execution

In most Microsoft debuggers, you can move the execution point to set the next statement of code to be executed.

Look for a yellow arrow in the margin. This marks the location of the next statement to be executed. By dragging the arrow, you can skip over a portion of code or return to a line previously executed.

dtSearch®

Instantly Search Terabytes of Data

across a desktop, network, Internet or Intranet site with dtSearch enterprise and developer products

Over 25 search features, with easy multicolor hit-highlighting options

dtSearch's document filters support popular file types, emails with multilevel attachments, databases, web data

Developers:
- APIs for .NET, Java and C++
- SDKs for Windows, UWP, Linux, Mac and Android
- See dtSearch.com for articles on faceted search, advanced data classification, working with SQL, NoSQL & other DBs, MS Azure, etc.

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

The Smart Choice for Text Retrieval® since 1991

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

Does Anybody Really Know What Time It Is: Dates and Times across Time Zones

The majority of the projects I've worked on in my career existed in a single time zone and used on-premises hardware. But how we handle systems that span time zones has changed and continues to change, and dealing with those changes is not always intuitive. As the cloud becomes more pervasive,

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.

it's becoming increasingly likely that, even if your client remains in only in a single time zone, their databases, services, and UIs may live in different time zones. And who knows, maybe someday your customer will expand or move. That's why I advocate building all new systems with the capability to handle multiple time zones.

In the past, not only did we not have a good way to handle multiple time zones, but each of the tools we used to build our systems had different support for dates, times, and time zones. In order to come up with a good solution, we often had to create multiple columns in our databases and multiple properties on our classes and keep them all in sync ourselves. A lot of developers wrote custom code to handle it and in some cases, we still have to do that today. But if you're a Microsoft stack developer, there are now pretty good ways to deal with this type of data built right into the tools.

The DateTimeOffset data type was originally introduced way back in .NET 2.0 at the end of 2005. Like its sibling the DateTime data type, it could store a date and/or time, but it added a third component, called the offset, which defaults to the current offset of the local time zone from Coordinated Universal Time (UTC it's not CUT because the acronym comes from the French language). For example, I live in Houston where the offset during Daylight Savings Time is -5 hours. That is, it's five hours earlier in Houston than UTC. UTC, sometimes referred to as Zulu because it's denoted with a "Z" next to the offset when written out, is a more precise standard than Greenwich Mean Time (GMT), which is the official time at the Royal Observatory in Greenwich, England. For the most part, you can think of UTC, Zulu, and GMT as basically the same thing. So when I say it's 1:41PM -5 here in Houston, it's the same as saying it's 6:41PM -0 UTC or 4:41AM +10 tomorrow morning in Canberra. When my services and database are in Azure (which uses the UTC time zone regardless of the data center's physical location) and timestamps something as 6:41PM (-0), I can see that that timestamp happened at 1:41PM local time (-5).

If you stored your data in SQL Server as many of us did, you had to wait three more years until SQL Server 2008 was released before you could write your .NET DateTimeOffset value into the database. Prior to that, you had to use one column to store the date and time and another to store the offset, and, if you were good, you knew enough to make sure your time offset column included decimal places so you could save offsets that weren't whole hours such as the +5:45 offset (5.75) used in parts of Europe and Africa. Most modern databases and programming languages support DateTimeOffset data types or an equivalent, so your best bet when writing a new system is to use DateTimeOffsets everywhere. When you do so, you record a moment in time that works across all time zones.

If it were only that easy! Although this strategy works for most scenarios, there are some interesting twists to be aware of.

If You Only Want to Store a Date

Although SQL Server also introduced data types that stored only a Date or only a Time in 2008, there are no matching data types in .NET. I've often felt this was an oversight. A mistake that many people make when storing only a date is using the DateTime type.

It's a mistake for several reasons: It's an especially dangerous situation because there's a quirk that only rears its head sometimes: when the time offset is great enough to change the time enough to cross midnight, which changes the date. In my scenario here in Houston, if I send my birthday as a DateTime of 9/29/1964 00:00:00 (midnight) to a service in Azure, it *might* subtract five hours (six in winter) and show up as 9/28/1964 making me a day older than I actually am! If I unintentionally send the current time along with the date, this bug shows up at different times of the day than in the cases where I'm careful to send the value as midnight.

Kind or Unkind?

You might be wondering why I said *might* subtract five hours (six in winter). That's an excellent question and one that baffled me for a while until I dug deep into the documentation of the .NET DateTime data type. The culprit is the Kind property, which was added to the DataTime structure in .NET 3.0. Kind is an enum that can be one of three values: Unspecified, Local, or UTC. The behavior we all seem to expect is when this property is set to the default value of DateTimeKind.Unspecified, which basically says to ignore any concept of an offset. If I specify 9/29/1964 00:00:00 in Houston and send it to Azure which is using UTC anywhere else in the world, the value is read as 9/29/1964 00:00:00. The documentation also says that

The Leading Independent Developer Magazine

CODE Magazine is an independent technology publication for today's software developers. CODE Magazine has been a trusted name among professional developers for more than 15 years, and publishes articles from more MVPs, Influencers and Gurus than any other industry magazine. Covering a wide range of technologies, our in-depth content is written by industry leaders; active developers with real-world coding experience in the topics they write about.

Get your free trial subscription at www.codemag.com/subscribe/free6ad

Helping Companies Build Better Software Since 1993

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

CODE MAGAZINE

you can use the DateTime.SpecifyKind() method to set this value. If you specified to use local time or UTC time, you can do that to and know what to expect.

What the documentation doesn't clearly specify is that some properties and methods on the DateTime type set the Kind property FOR YOU.

What the documentation doesn't clearly specify is that some properties and methods on the DateTime type set the Kind property FOR YOU. You can only find that documented in those methods. For example, if I wrote this bit of code:

```
var n = DateTime.Today;
```

You see that the Kind property is set to Local, but if I run this code:

```
var n = new DateTime(n.Year, n.Month, n.Day);
```

You see that the Kind property is set to Unspecified.

In the debugger, these DateTime values look identical except for the **Kind** property which few people know to check. If I send both of these values from my app running in Houston to a service in Azure to be stored in a database, I *might* get different results when I ask for the data back. The value in the first example will likely be adjusted by -5 hours. If I store the value in the database in a DateTime column, then it will be five hours off and if I store it in a Date column, it will be stored as yesterday.

Why? What's the difference between these two examples? When you use things like DateTime.Now or DateTime.Today, the Kind property is set to Local to indicate that it was set from the computer's local clock. However, when you create a DateTime with explicit values for year, month, and day, the local clock isn't used (n.Year, n.Month, and n.Day are just integers in this context) and the Kind property is set to Unspecified.

The bottom line is, use DateTimeOffsets when you can. They save a lot of headaches, even if you don't care about the offset. Alternatively, you can create your own .NET type to wrap a DateTime so that the Kind property is always set to Unspecified whenever a value is set. Although that smells a bit like a hack, it works and you may be stuck with it on some projects.

If You Only Want to Store a Time

As a .NET developer, you have a couple of choices. The TimeSpan class is very handy for manipulating hours, minutes, seconds, or ticks, and it maps relatively well to SQL Server's Time data type. Alternately, you can store the hours, minutes, or seconds since midnight in an int in both .NET and SQL Server or, if you need more precision, you can store ticks since midnight as a four-byte integer (a long in .NET and a bigint in SQL Server).

A Word about Time Zones

Time zones and offsets are not the same thing. One problem that DateTimeOffsets don't solve is storing the originating time zone. Knowing the offset isn't enough to tell you which time zone it came from. For instance, Arizona doesn't observe Daylight Savings Time, so in summer it's the same time in California as it is in Arizona, but in the winter, it's not. Also, there are plenty of instances where time zones overlap. If you need to store data with time zones or convert DateTimeOffsets to a particular time zone, see .NET's TimeZoneInformation class, introduced in version 3.5.

An Unusual Cases

There will always be a few cases that don't fit the mold. For example, I recently worked with a client who scheduled appointments for their customers. To them, a 10AM appointment on July 1 should show up as a 10:00 AM appointment on July 1, no matter which time zone it's viewed in. If an associate in a branch in London looks at a schedule for a branch in Los Angeles, they don't want to see that appointment showing up as 2AM (which is what time it would actually take place). In cases like this, it's best to store a DateTimeOffset along with TimeZoneInfo, and use the DateTimeOffset.ToOffset() method to get whichever version of the appointment time is required for the situation. Alternatively, you could use a DateTime and ensure that the Kind property is set to Unspecified. This has the disadvantage of never knowing exactly when something actually happened. In this scenario, you'll just never know exactly when a 5PM appointment starts, but hey, it's 5 o'clock somewhere!

Summary

I hope this has cleared up at least few questions for you and maybe even got you thinking a bit! Dates and times are intuitive, but actually quite difficult. Time zones, daylight savings time, leap years, leap seconds, and even different calendars all come in to play. Fortunately, .NET gives us a pretty good arsenal of tools to work with to handle the various situations. My wish list includes adding Date and Time data types to .NET and introducing a DateTime-like type that ignores offsets so that it works consistently. I don't think it's going to happen, but I'll give it some time.

Mike Yeager

(Continued from 74)

a whole (such as our highly broken interview processes). Ditto for this column. My Twitter account, being more a reflection of me as a person, I allow to show my political views on topics and when I feel strongly on a subject. My Facebook account, which I keep invite-only, is generally all-personal-no-professional. I don't suggest that this is the "best" way to use social media; this is simply the plan that I've evolved. You need to make your own plan, so that you can be clear about where your opinions will go and who they will reach. Which brings us back around to the other part of this.

A High Degree of Discipline

Having made that clear delineation in your head, the software development professional now faces a truly hard task: sticking to that plan. This is where a high degree of discipline is required: You must stick to your plan, religiously, despite the myriad opportunities that exist to bring the "wisdom" of your words to the masses around you through the various channels open to you. Stick to your plan. Bring that wisdom only through the channels that won't impede your professionalism.

Without rendering judgment on the Google employee's remarks, we can still discover how he failed in his professional duties. As near as we can tell, his opinions were written at work using work resources and time, meaning they were essentially a work product; once they became detrimental to the company's health, he crossed over into the realm of "poor judgment." This is no different than if he were to use his work resources to write an opinion piece about how his employer's competitors engage in slave labor or outright intellectual property theft; the actual subject is irrelevant to the discussion. Although some people might debate that this is an issue of "free speech," the larger point is that he put his employer into an uncomfortable position, and one which garnered some extremely bad press. Satya Nadella made a similar mistake in the early days of his tenure at Microsoft. One of these two individuals immediately apologized and sought to right the wrong, and remains employed to this day; one of them refused, was terminated, and is currently suing his former employer.

As software development professionals, if we are to claim that term, we have to understand that our professionalism is also built around the perception others have of us as we carry out our duties. I most certainly encourage every software developer to be as engaged in their political community of choice as they choose to be, but understand that there is a clear demarcation between your personal world and your "professional" one, particularly for those of us who build a personal brand. If you choose to bring your personal opinions with you into the workplace, you also choose to accept the consequences of doing so. Right or wrong.

Ted Neward

CODE COMPILERS

Nov/Dec 2017
Volume 18 Issue 6

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, Vassili Kaplan
Wei-Meng Lee, Sahil Malik
Jeff Palermo, Ted Neward
John V. Petersen, Nic Raboy
Justin Self, Paul D. Sheriff
Chris G. Williams, Mike Yeager

Technical Reviewers
Markus Egger
Rod Paddock

Art & Layout
King Laurin GmbH
info@raffeiner.bz.it

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 Professionalism

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. As I write this, a well-known software prognosticator and talking head finds that his website is being DDoS-attacked by what he's calling "Social Justice Fascists." He believes they're doing it in reaction to a blog post that he wrote that expresses a strongly conservative-leaning opinion about a variety of topics related to the technology industry's responsibility around hiring, firing, conference invites, and so on. As of this writing, in fact, his four most recent blog posts are about topics that are more appropriate to a political magazine than a technical blog.

A few months ago, a Google employee wrote an informal memo to his coworkers about what he saw as misguided attempts on Google's part to allocate resources (time, energy, and money, among other things) to a cause that he believed was inefficient and a waste. His superiors disagreed and terminated his position. He's in the process of suing them for wrongful termination. The industry, meanwhile, exploded into debate over the topic.

This is not an article about that topic.

In an article that appears in this issue of CODE magazine, John Petersen talks about the ethics of professional programmer conduct, and whether or not the software industry should be regulated. He notes that development is currently not a "profession" in the same way that professional engineers (those who build buildings and bridges and such) are defined, and that if we seek to create an organization that upholds professional standards, we're asking for regulation, and that we should be careful what we ask for. In particular, John notes:

It's ironic that many software developers refer to themselves as software engineers. ... If you want to hold yourself to a higher standard, be professional, conduct yourself accordingly, and be accountable. At the end of the day, that's what being a professional is all about.

The author of the blog posts suffering the DDoS attacks has historically been the first to suggest that we as developers need to hold ourselves to a high standard of professionalism, that we should see ourselves as "software craftsmen."

There's a thread here, loosely tied, bringing these two topics together: Professionalism is a harder thing to accomplish than we might believe. It requires a clear decision and a high degree of discipline.

A Clear Decision

Professionalism is defined by Merriam-Webster online as "the skill, good judgment, and polite behavior that is expected from a person who is trained to do a job well." (See http://www.learnersdictionary.com/definition/professionalism.) It's important to note that the second and third elements in that definition are not related to the skills but the manner in which one conducts oneself. "Good judgment" is, of course, a relative term, but "polite behavior" is pretty clear in casual use; contrast that with the traditional archetype of the "arrogant genius" as embodied by TV heroes such as Dr. Gregory House, who routinely abuses his coworkers and staff, yet every episode demonstrates that he's "the best damn doctor in this hospital" by saving people who would've died without his brilliant insights and encyclopedic knowledge of human affliction. Is House a great doctor? Maybe. Is he a professional? Certainly not by the above definition.

Cue to John's discussion of codes of conduct, such as those listed and described at software conferences: "Such codes... are an attempt to provide an objective standard of conduct. Put another way, these codes are meant to be objective yardsticks by which to judge behavior." The medical and legal professions have, for decades, established what is and is not considered to be ethical behavior regarding the manner with which doctors and lawyers are allowed to interact with their clients. (House spends a great deal of each episode fighting with his boss and his staff about working around those guidelines.) Lacking those explicit guidelines, however vague or precise, we're left with an interesting and important decision: Given that the Western world permits citizens the right to speak their minds (to a greater or lesser degree, depending on the context), how professional is it to air your thoughts on a given subject in a professional workplace?

I have my own opinions on this subject. What matters more, however, is what degree you, dear reader, believe is appropriate. Because that's the decision you have to make: If you take a stand for what you believe in, and state it loudly enough, you may find yourself the target of people who disagree. This isn't even a political discussion: Over a decade ago, I wrote the essay "Object/Relational Mapping is the Vietnam of Computer Science," which gathered a firestorm of technical debate that still rages to this day. Those who advocate the use of such tools (like Entity Framework) blasted me for my ignorance; those who had suffered the inefficiencies of such tools sang my praises. I was alternatively vilified and canonized.

When you share your opinion on a subject, you run the risk of attracting attention both good and bad, largely dependent on the degree of your position and the size of your audience.

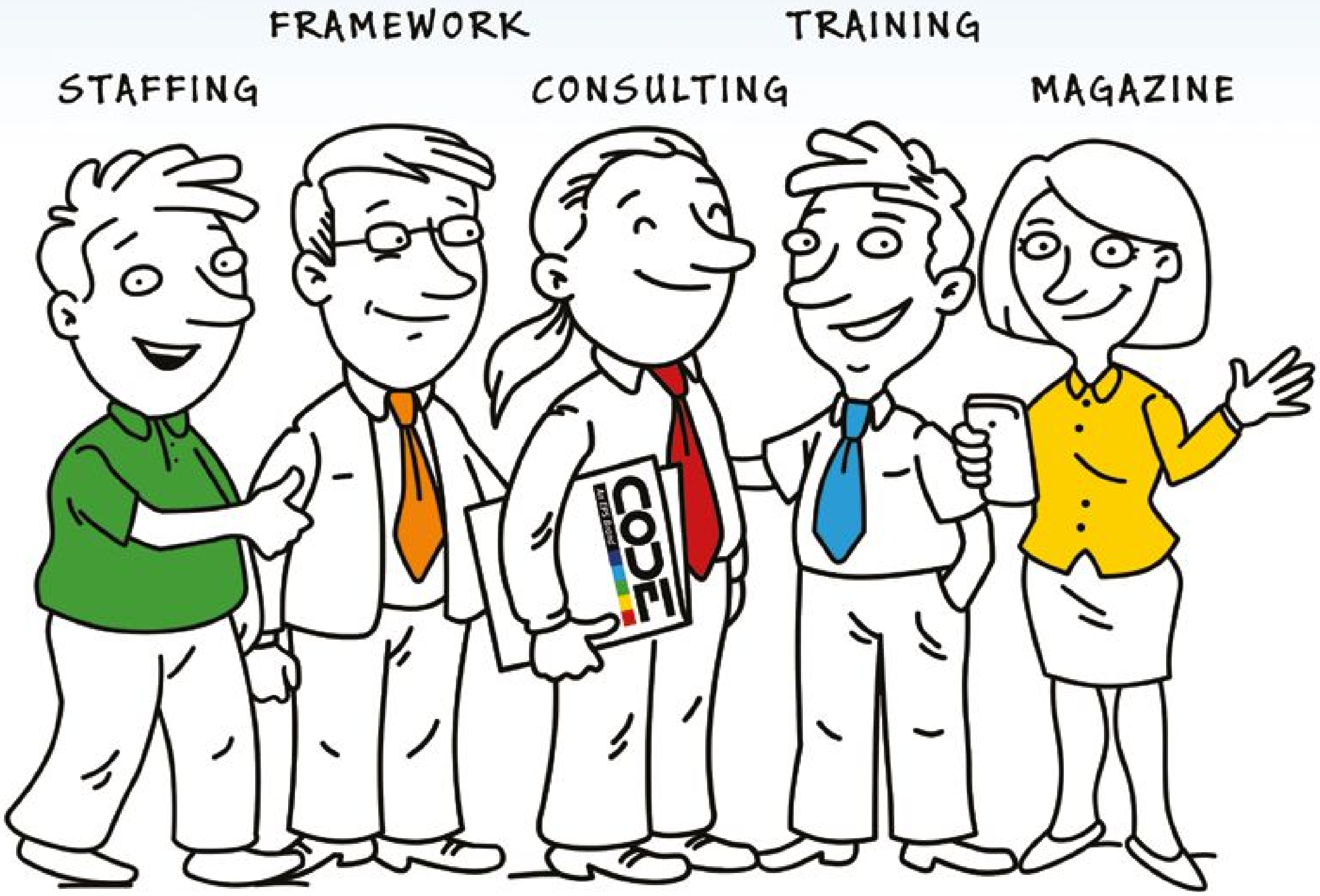
Here's the decision: To what degree will you, as a software development professional, put your personal opinion out there onto the Internet? To refuse to share your thoughts is going to impede your technical progress, to be sure. But to share your thoughts runs the risk of stepping into a firestorm of controversy, however large or small.

But more to the original point, what topics will you weigh in on? Do you wish to take a stand on topics outside of the technical arena, or do you wish to keep your discussions more industry-focused? This is where we come back to the professionalism question: If the definition of professionalism is to use "good judgment" and "polite behavior," you may find that certain topics inherently work against those two elements. You need to be absolutely clear in your own mind where your lines are on this subject, largely so that you can be fully ready to reap the benefits and costs of making those choices.

Please note: You don't need to blanket your position across all media. Myself, I choose to have an entirely industry-focused blog, although I do give myself the freedom to weigh in with strong opinions about the industry as

(Continued on page 73)

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

RD

Rider

New .NET IDE

Cross-platform.
Killer code analysis.
Great for refactoring.

From the makers of ReSharper, IntelliJ IDEA, and WebStorm.

Learn more
and download
jetbrains.com/rider

JET
BRAINS