iMessages, F# Type Providers, Swagger, Cordova

CODE

MAR APR 2017

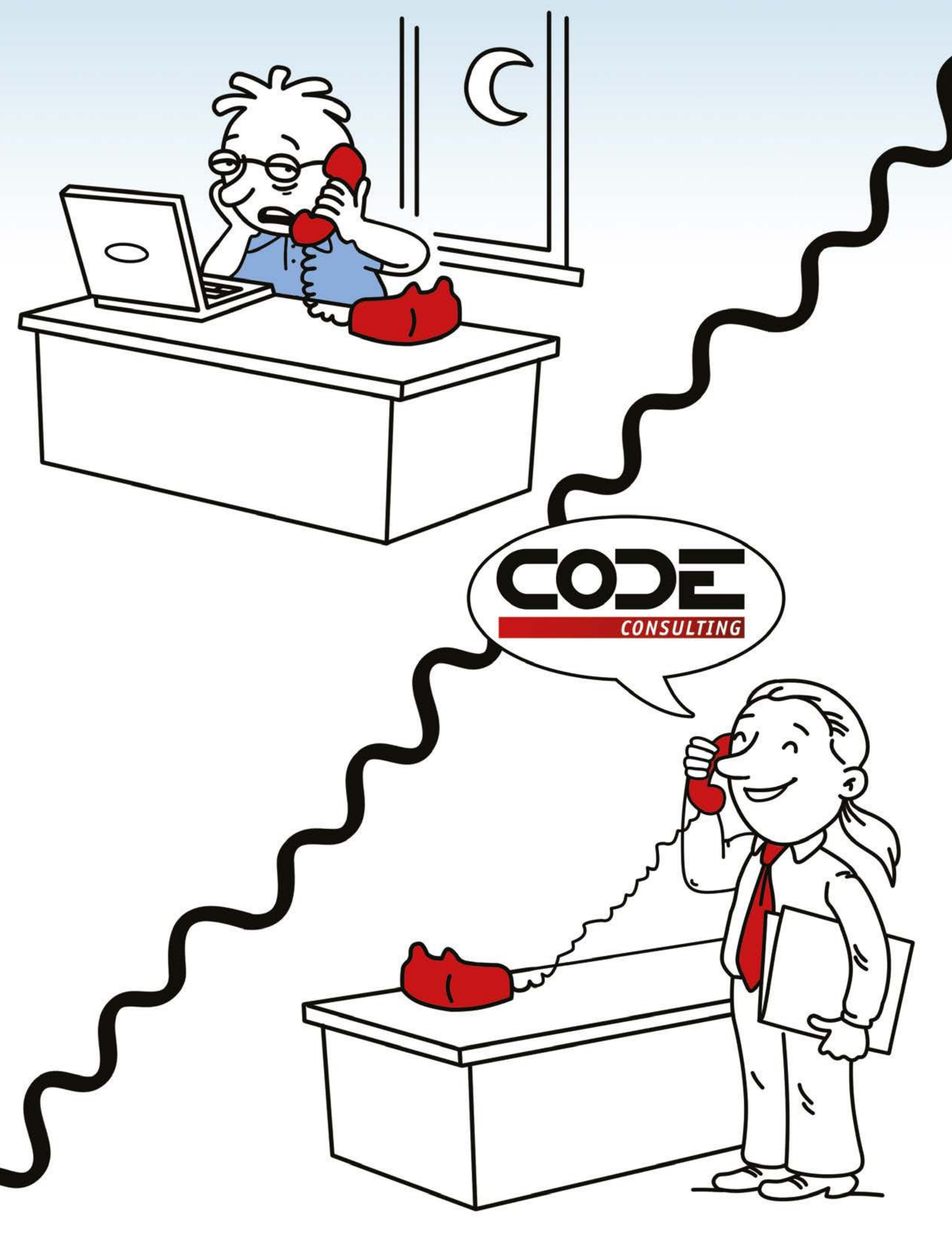
codemag.com - THE LEADING INDEPENDENT DEVELOPER MAGAZINE - US $ 5.95 Can $ 8.95

Programming Alexa Skills for the Amazon Echo

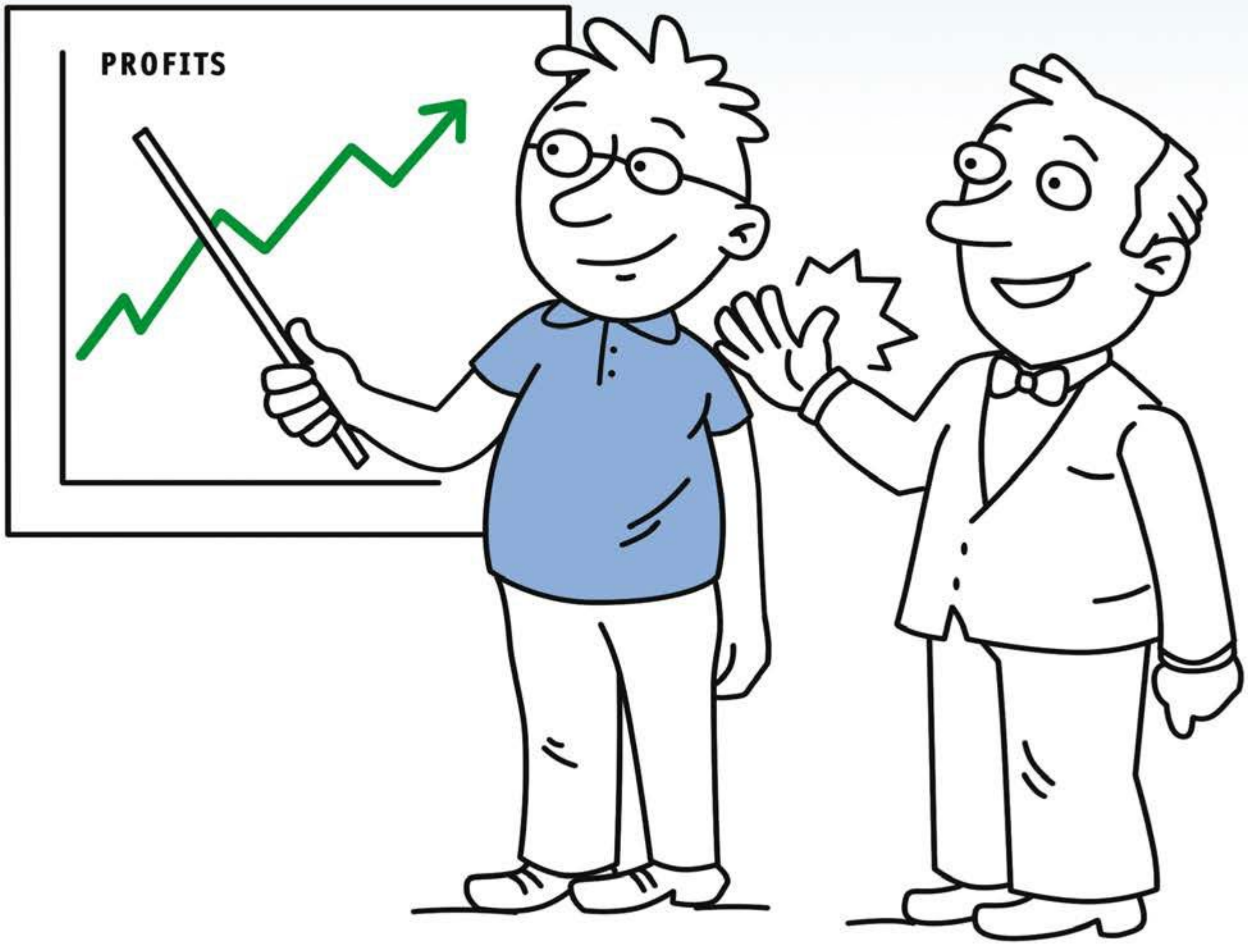
Xamarin versus Cordova

Azure Skyline and Docker

Accessing Data with F# Type Providers


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

CODE CONSULTING

Features

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 or contact customer service at 832-717-4445 ext 028.

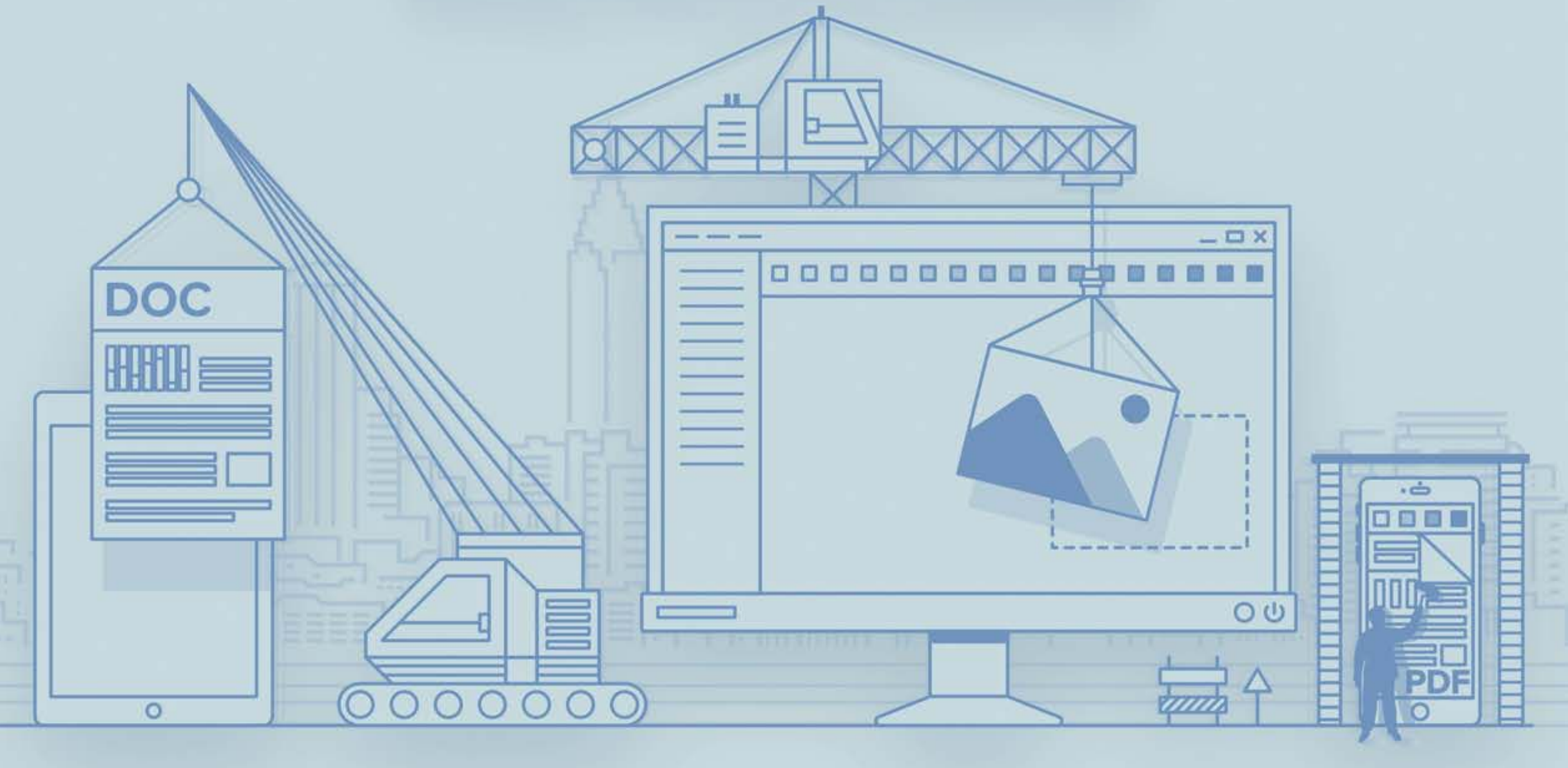
Subscribe online at codemag.com

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



BUILD BETTER APPS WITH LEADTOOLS

DOCUMENT

Document Viewer & Converter

OCR, MICR, OMR & ICR

1D & 2D Barcode

Forms Recognition

PDF, DOCX, HTML, SVG, RTF, TXT

Annotations & TWAIN

MEDICAL

DICOM, CCOW & HL7

PACS Client & Server Framework

DICOMWeb (WADO)

Web & Desktop Viewers

Image Processing & Annotations

Medical 3D (MPR, MIP, VRT)

MULTIMEDIA

Play, Capture, Convert & DVR

Media Streaming

MPEG-2 TS, RTSP, HTML5 & more

OGG, FLV, ISO, AVI, WebM & more

Audio & Video Processing

DirectShow & Media Foundation

.NET WinRT Windows API Java Linux iOS macOS Android JavaScript

DOWNLOAD OUR 60 DAY EVALUATION
WWW.LEADTOOLS.COM

LEAD TECHNOLOGIES INCORPORATED

SALES@LEADTOOLS.COM
800.637.1840

You're Not Finished Yet

I've edited a whole lot of books, articles, help files, websites, and white papers about technology. I've seen countless products come and go and I've been on plenty of teams building apps or software. At the end of the day, there's one thing I'm sure of: Everything needs to be tested before it's sent out the door, whether it's the software, the device, or the words. Not only will your product (or article, etc.) be better, but you'll grow, too.

Editing text is different from testing software, but it's the same principle. You want users to effortlessly read the words and use the product. You need to know if the dialog box you see as the gateway to your product contains a mystifying set of text boxes or if the search function produces random results. You don't want people to struggle to use your product; you want a clear flow of information or function, and clear help files in case they get stuck.

I once worked at ZD Labs, which supported all 12 of Ziff-Davis Publishing's tech magazines. Our group took hardware and software and did our best to break it. We built devices that held laptops precariously over various surfaces and dropped them repeatedly from a specific and controlled height. We wired keyboards and spilled various substances (soda, water, coffee, etc.) on them until they failed, and measured how much electricity zapped the keyboard and whether it went with a whimper or a bang. We did the opposite of what every well-intentioned instruction told us to do. We had a Faraday Cage to deliberately electrocute computers and a completely soundproof room where we could measure how loud that fan keeping your hard drive cool really was or how much noise a room full of mainframes made. In short, we had a whole lot of fun.

The object wasn't really to have fun but to use the product in the worst possible way in a controlled environment so that the various tragedies could be measured. We barely read, entirely skipped, or obstinately misinterpreted instructions. We researched statistics—how many seated users knocked a laptop off their laps or dropped one while running through an airport. We even had a rubber finger on a stick that poked the same ke, around the clock for weeks on end until the key broke.

We studied whether a non-technical person could figure out how to use a piece of software. To perform such a test, I built a database using FoxPro that tracked the timecards and contract lengths for ZD Labs' team of freelancers. I enjoyed it enough that I next volunteered to build a much more involved database using an Oracle product (with a lot of help) to set up a software and literature library at the lab.

The whole reason for ZD Labs' existence was to find out whether products kept the promises on their boxes, and to see if a reasonable amount of use or abuse might break them. We set benchmarks to compare similar products or functions, and we brought in freelancers with widely varied skill sets to see how much knowledge was needed to succeed—or fail. Our purpose wasn't to help the manufacturers, although that was probably a result. We were trying to inform magazine readers so they could make intelligent purchases and use the products in an advantageous way.

No matter how brilliant you are as a software designer, you need someone to be your first user who can provide the kind of feedback that helps you deliver a great product before it goes public. It's the same if you're writing about technology or providing the supporting documents that ship with the product. You can use (or reread) your work again and again, but you still know what the original intention was, and logical lapses won't jump out at you the way they would to someone who's not using your brain. You need someone to intentionally break your product and then give you feedback.

What if your first user (or editor) has different taste or disagrees with your design? What if they don't understand your goal, or worse, what if they inject their own ideas into your work? That's the second half of having your work tested or edited. As important as it is to get feedback, it's just as important to know how to respond to it.

Let's say that you build an app that measures hiking paths. You track whether the paths are also suitable for horses, bicycles, or small motor vehicles, you've got elevation change, and you track distance and give a letter grade for difficulty. Your beta user wants a bigger picture—parking availability, running and hiking clubs that use the park, handicapped access, bathroom and water availability, and so forth.

Those are good ideas, of course, but perhaps that's not the app you're building. Or perhaps those are on the wish list (backlog) for V2, but you've got a deadline to meet. It's your job to figure out whether the app's users need those things or whether they're just bells and whistles. You might have to do a cost analysis to make this decision or maybe you already know the right thing to do.

Whether you rush to add those features or not, it's important to consider that your tester (editor) won't be the only person to have those thoughts. They're merely the first. You need to consider the cost of delivering something that might easily be seen as incomplete, even if you disagree.

I've seen more unpolished text than software as an editor, but it works the same way. If you're writing the help files or UI, or even a nice article, it's important that your first reader understands exactly what you mean. Instructions are not the time to get fancy or play with hyperbole or other literary devices. Oh, you're welcome to be humorous, clever, or even charming (if you are those things), but not to the point that the reader thinks about YOU rather than the product.

A good tester or editor will know what you meant and be able to help you produce the result that you had in mind. You might even be a pretty good tester or editor yourself. But there's a danger in thinking that you can be your own tester or editor. You know that old adage, "The lawyer who represents himself has a fool for a client," right?

A good dev or writer can certainly make a decent pass at his own work, tweaking here and tidying there. And that should happen, every time, whether you're writing code or text. But it's a very bad idea to deliver a product that only you have tested or edited.

It's the weird interpretations other people make that allow those flashes of inspiration and insight and make your work better. That's how you grow. You'll never be finished growing if you're good at what you do.

Post Script: One of my editor friends suggested a Dr. Dobb's Journal frequent blogger's posts about testing. The Dr. Dobb's site went dark in 2014, but the articles, and many new ones by Michael Hunter, are available here: http://www.thebraidytester.com/. I've got more than a hundred articles on writing for technical people on my website, at www.MelanieSpiller.com.

Melanie Spiller

DOMAINS | MAIL | HOSTING | eCOMMERCE | **SERVERS**

STOP SHARING!

1&1 VIRTUAL SERVER CLOUD

starting at **$4.99** per month*

Trusted Performance.
Intel® Xeon® processors.

Windows

NEW

1&1 eliminates the "noisy neighbor effect": Ideal for beginners as a web and mail server, but also for more demanding projects like database applications, the new 1&1 Virtual Server Cloud is 100% yours to use! Take advantage now of the latest cloud technology.

- No shared resources through VMware virtualization
- Full root access
- SSD storage
- Unlimited traffic
- High performance
- Maximum security
- Best price-performance ratio
- 24/7 expert support
- Choice between Linux/Windows
- Plesk ONYX

1 CALL SPEAK WITH AN EXPERT 24/7

1 MONTH FLEXIBLE PAYMENT OPTIONS

1 SECURE CHOICE FAIL-SAFE

1-844-296-2059

1&1®

1and1.com

* 1&1 Virtual Server Cloud S: $4.99/month. Billing cycle 1 month. Minimum contract term 12 months. No set up fee. © 1&1 Internet Inc. 2017 All rights reserved. 1&1 and the 1&1 logo are trademarks of 1&1 Internet SE, all other trademarks are the property of their respective owners. Windows and the Windows logo are registered trademarks of Microsoft Corporation in the United States and/or other countries. 1&1 Internet Inc, 701 Lee Road, Chesterbrook, PA 19087.

Data-Driven Testing with Visual Studio

Every developer needs to test their own code or have it tested by someone else. Many developers aren't great at testing their own work. The main reason is that we tend to test only the "happy path" through the functionality that we wrote. We often avoid testing the boundaries of our code, such as invalid inputs, exceptions that might occur, etc. One way to become a better

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

Paul D. Sheriff is the President of PDSA, Inc. PDSA develops custom business applications specializing in Web and mobile technologies. PDSA, founded in 1991, has successfully delivered advanced custom application software to a wide range of customers and diverse industries. With a team of dedicated experts, PDSA delivers cost-effective solutions, on-time and on-budget, using innovative tools and processes to better manage today's complex and competitive environment.
Paul is also a Pluralsight author. Check out his videos at http://www.pluralsight.com/author/paul-sheriff.

tester is to start writing unit tests. Although it takes more time up-front to write unit tests, it saves a ton of time if you have to regress test changes to existing features.

Starting with Visual Studio 2008, Microsoft added a unit testing framework into Visual Studio. One of the great features of the unit testing tools in Visual Studio is the ability to use a data-driven approach. There are also several third-party testing frameworks you can use. In this article, you'll to learn to retrieve values to be used for testing from SQL Server. Using a data-driven approach can significantly reduce the unit tests you must write.

Why You Should Test Your Code

We all know that we need to test code prior to putting it into production. Testing ensures that the code you write works as expected. It's not enough to check that it works as expected, but you must also check it under varying circumstances and with different inputs. For example, what if someone takes the database offline? Will your code recover gracefully from this exception? Does your code inform the user that there's a problem with the database connection in a friendly and easily understood manner? All of these questions need to be answered in the testing of your code. You need to simulate these conditions so you can test your exception code.

Performing the act of testing often helps you improve the quality of your code. As you think about the various scenarios that can go wrong, you start to add additional code to handle these scenarios. This leads to your code being more robust and ultimately more maintainable and user-friendly. You'll find that taking time to test your code will make you a better programmer in the long run. Your boss and your end-users will appreciate the extra effort as well.

Automated Testing

Instead of you or a QA person doing all the testing, try to automate as much as possible, called unit testing. Unit testing means that you write a program, or more likely, a series of programs, to test each line of code and ensure that it operates properly under all circumstances. Unit testing has become very prevalent in today's professional software shops.

Although this sounds like a lot of work, and it *is* more work up front, you'll more than make up that time by avoiding multiple regression tests that have to be done by a human. You can even automate the set-up and tear-down of databases, files, etc. With the correct architecture, you can automate almost all of the various inputs a human tester would have to enter by hand.

You also save the time that you normally eat up when you do your own testing by pressing F5 to start debugging in Visual Studio, waiting for the compile process, and clicking through a half dozen menus to finally get to the point where you want to start testing. As you know, this can sometimes take 15 seconds to a minute, depending on how large your application is. By unit testing instead, you can run many tests in just a couple of seconds. This adds up to saving many hours over your complete development cycle.

Automated tests are repeatable, unlike tests run by humans who might forget to test something. You end up with more of your code tested because things won't be forgotten. Because you're forced to think of how to test your code while you're writing it, you write better code. You also save time on the set-up and the tear down of the tests because these tasks can also be automated. As you can see, there are many advantages to an automated approach over human testing.

Of course, there are disadvantages to the automated approach as well. First, it does take more time up-front to develop these tests. Automated tests are only as good as the person who develops the tests. The tools to test user interfaces are not always great, and they require that you purchase third-party testing tools. Then again, if you're using good n-tier techniques, MVC, MVP, MVVM, or similar design patterns, there should be very little UI for you to test anyway.

The FileExists Method

For this article, you're going to build a method that checks to see whether a file exists. You're then going to build unit tests to check each type of input that you can pass to this method. Next, you're going to replace all of those unit tests with a single unit test that retrieves the various inputs from a database table.

To start, create a new Class Library project in Visual Studio using C#. Set the name of this class library project to MyClasses. Rename the Class1.cs file created by Visual Studio to FileProcess. Add a method in this class called FileExists, as shown in the following code snippet:

```
public bool FileExists(string fileName) {
  if (string.IsNullOrEmpty(fileName)) {
    throw new ArgumentNullException("fileName");
  }

  return File.Exists(fileName);
}
```

This is a very simple method, yet it requires at least three unit test methods to ensure that this method works with all possible inputs. The three possible values you can pass to the fileName parameter are:

- A file name that exists
- A file name that does not exist
- A null or empty string

Create a Test Project

Right-mouse click on your MyClasses solution and choose **Add** -> **New Project**. From the list of templates, click on the **Visual C#** -> **Test** -> **Unit Test Project**. Set the **Name** to **MyClassesTest**. Click the **OK** button. Rename the **UnitTest1.cs** file to **FileProcessTest.cs**. You're going to test the method in your MyClasses class library, so you need to add a reference to that project. Right-mouse click on the References folder in the MyClassesTest project and select MyClasses. Add the following using statement at the top of the FileProcessTest.cs file.

```
using MyClasses;
```

It's now time to start writing the unit tests to create each of the three possible inputs identified for this method. The first one is to test that a file exists. Add the code shown below to the FileProcessTest class. Feel free to change the drive letter, path, and file name to a file that exists on your computer.

```
[TestMethod]
public void FileExistsTestTrue() {
  FileProcess fp = new FileProcess();
  bool fromCall;

  fromCall =
    fp.FileExists(@"C:\Windows\Regedit.exe");

  Assert.AreEqual(true, fromCall);
}
```

After adding this code, right-mouse click in your code window and choose **Run Tests** from the context-sensitive menu that appears. After the code runs, a Test Explorer window appears with the results of the test. If the file exists, the window should display something that looks like **Figure 1**.

The next method to write tests for a file that doesn't exist. Create a method in your FileProcessTest class to test this condition. Write the code shown in the following code snippet.

```
[TestMethod]
public void FileExistsTestFalse() {
  FileProcess fp = new FileProcess();
  bool fromCall;

  fromCall =
    fp.FileExists(@"C:\BadFileName.txt");

  Assert.AreEqual(false, fromCall);
}
```

Once again, run these tests and you should now see two passed tests in your Test Explorer window. Add the final test to the FileProcessTest class to test if you pass in a null value or a blank value to the FileExists method. An ArgumentNullException is thrown from the FileExists method if a null or blank value is passed to the method. There are two ways to handle this thrown exception. First, add an ExpectedException attribute after the TestMethod attribute on the method, as shown in the following code snippet.

```
[TestMethod]
[ExpectedException(typeof(ArgumentNullException))]
public void FileExistsTestNullWithAttribute() {
  FileProcess fp = new FileProcess();

  fp.FileExists("");
}
```

The second way to handle this exception is to wrap up the call to the FileExists method within a try...catch block. The catch block should check to see if the exception is an ArgumentNullException. If it is, don't do anything to tell the unit test framework that this test succeeded. If no exception is thrown, there's something wrong with the logic and you should call the Assert.Fail() method to signal to the unit test framework that this test failed. Let's use this second method because this is what you'll do when you go to a data-driven approach.

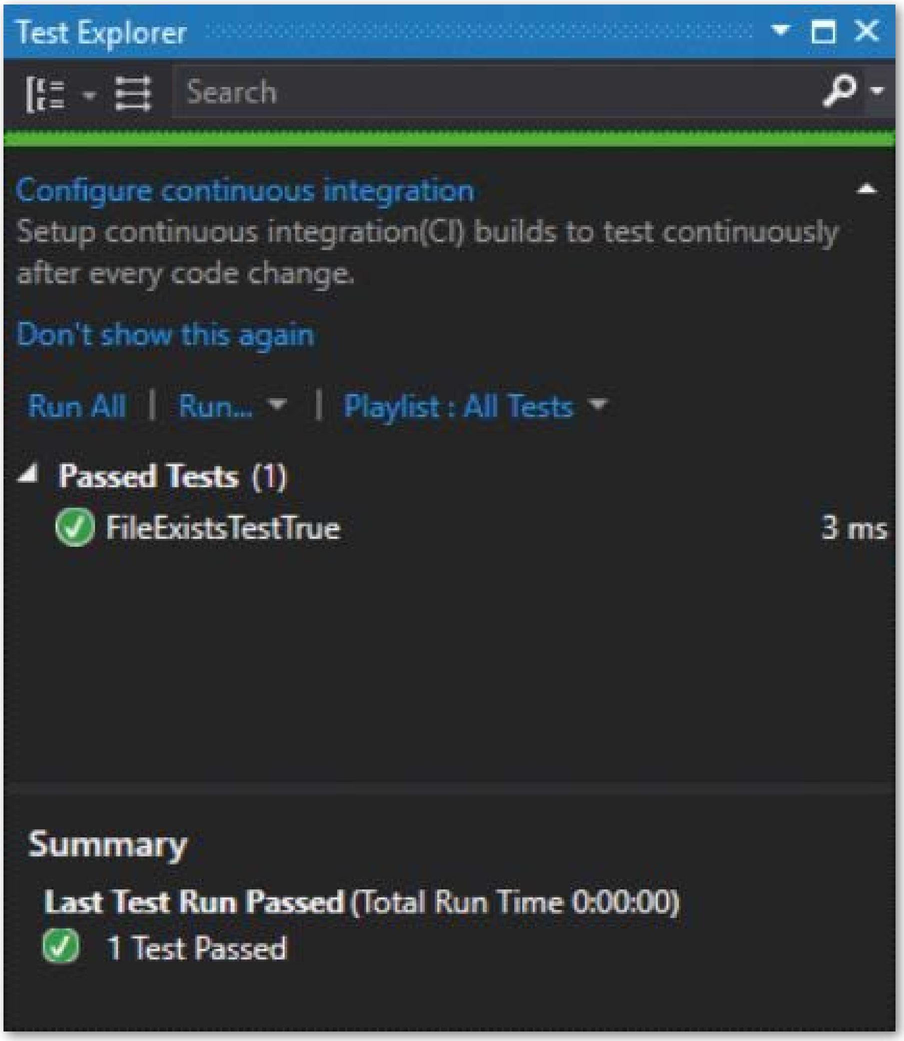

Test Explorer
Search
Configure continuous integration
Setup continuous integration(CI) builds to test continuously after every code change.
Don't show this again
Run All | Run... | Playlist : All Tests
Passed Tests (1)
FileExistsTestTrue 3 ms
Summary
Last Test Run Passed (Total Run Time 0:00:00)
1 Test Passed

Figure 1: Test results appear in the Test Explorer window.

```
[TestMethod]
public void FileExistsTestNull() {
  FileProcess fp = new FileProcess();

  try {
    fp.FileExists("");
    Assert.Fail();
  }
  catch (ArgumentNullException) {
    // The test succeeded
  }
}
```

Run the unit tests one more time, and you should now see three passed tests in the Test Explorer window. Using this approach to testing means that you need to create different methods for each individual file and return value combination that you wish to test. Extrapolate this to a method that has several parameters, multiple If statements or Switch statements, and you can see how the number of methods you must write can grow substantially. This is where a data-driven approach to passing parameters to your methods reduces the amount of test methods you need to create.

Create a Data-Driven Test

Now that you have a table, let's create a test method that can be fed data from the FileProcessTest table you created previously. There are a few steps to get the data-driven test working.

1. Create a table to hold parameter data.
2. Add a reference to System.Data to your test project.
3. Add a TestContext property to your test class.
4. Add a DataSource attribute to your test method.
5. Add a config file to hold a connection string and table name to use.

Create a Table to Hold Test Data

Because you created three test methods, you're going to need a table with three rows, one for each test. Each test passes one value, and returns a Boolean value, so the table needs at least two fields: FileName and ReturnValue. Add one more column named CausesException that you can set to a true or false value on whether you are expecting an exception to be thrown by the method you are testing. Create this table in a database with the script shown in **Listing 1**.

The data in this table that was created from the script matches the hard-coded values in the three-unit test methods. **Table 1** shows the values that are now in your table. Feel free to use any specific file names and return values that make sense for your computer.

Add Properties and References

When the unit test framework creates an instance of a test class (a class marked with a [TestClases] attribute), it creates a TestContext object. This object contains properties and methods related to testing. One of the properties stores the data source information to allow you to access each row in the table you specify for the test. To access this TestContext object, you must create a property named TestContext in each of your test classes.

```
private TestContext _TestInstance;
public TestContext TestContext
{
    get { return _TestInstance; }
    set { _TestInstance = value; }
}
```

A DataTable is created from the table you create and a DataRow property is exposed on your TestContext property. In order to use this property, you must add a reference to the System.Data.dll. Go ahead and right-mouse click on your test project's References folder and select Add Reference… from the menu. Then select **Assemblies -> Framework,** locate the System.Data assembly, and add it to your project.

Create Data-Driven Test Method

Add a new method to your FileProcessTest class that looks like the following code snippet. Don't split the connection string across multiple lines, though; I had to format it to fit within the column width of this article. You also need to modify the name of your server and the database in which you created the FileProcessTest table.

```
[TestMethod()]
[DataSource("System.Data.SqlClient",
  "Server=Localhost;
   Database=Sandbox;
   Integrated Security=SSPI",
  "tests.FileProcessTest",
  DataAccessMethod.Sequential)]
public void FileExistsTestFromDB() {

}
```

Listing 1: Create a table to hold test values

```
CREATE SCHEMA tests
GO

CREATE TABLE tests.FileProcessTest
(
         FileName varchar(255) NULL,
         ExpectedValue [bit] NOT NULL,
         CausesException [bit] NOT NULL
)
GO

INSERT INTO tests.FileProcessTest
VALUES ('D:\Exists.txt', 1, 0);

INSERT INTO tests.FileProcessTest
VALUES ('D:\NotExists.txt', 0, 0);

INSERT INTO tests.FileProcessTest
VALUES (null, 0, 1);
```

FileName	ReturnValue	CausesException
C:\Windows\Regedit.exe	true	false
C:\NotExists.xls	false	false
NULL	false	true

Table 1: Data for the FileProcessTest table

Besides the [TestMethod] attribute, this method has a new attribute called [DataSource]. This attribute allows you to specify a data provider, a connection string, the name of the table in the data source, and how to loop through the records in the table.

The unit testing framework uses the information passed to the DataSource attribute to build a DataTable of the rows within the table specified in the DataSource. The framework begins looping through each row and for each row, sets the DataRow property and then calls your test method. You now retrieve the appropriate columns you need for your test within your method. **Listing 2** shows the complete code for the FileExistsTestFromDB() method.

In the FileExistsTestFromDB method, you retrieve the values from the column using standard ADO.NET syntax. The next code snippet shows retrieving each of the columns you created in the table.

```
fileName = TestContext.DataRow["FileName"]
   .ToString();
```

Listing 2: A data-driven test method

```
[TestMethod()]
[DataSource("System.Data.SqlClient",
  "Server=Localhost;
   Database=Sandbox;
   Integrated Security=SSPI",
  "tests.FileProcessTest",
  DataAccessMethod.Sequential)]
public void FileExistsTestFromDB() {
  FileProcess fp = new FileProcess();
  string fileName;
  bool returnValue;
  bool causesException;
  bool fromCall;

  // Get values from data row
  fileName = TestContext.DataRow["FileName"]
     .ToString();
  returnValue = Convert.ToBoolean(
     TestContext.DataRow["ReturnValue"]);
  causesException = Convert.ToBoolean(
     TestContext.DataRow["CausesException"]);

  // Make call to Method
  fromCall = fp.FileExists(fileName);

  // Check assertion
  try {
    Assert.AreEqual(returnValue, fromCall,
      "File Name: " + fileName +
      " has failed it's existence test in test:
        FileExistsTestFromDB()");
  }
  catch (AssertFailedException ex) {
    // Rethrow assertion
    throw ex;
  }
  catch (Exception) {
    // See if method was expected
    // to throw an exception
    Assert.IsTrue(causesException);
  }
}
```

Advertisers Index

ADVERTISERS INDEX

1&1 Internet, Inc.	www.1and1.com	7
AnDevCon	www.andevcon.com	67
CODE Consulting	www.codemag.com/techhelp	2, 37
CODE Divisions	www.codemag.com	76
CODE Framework	www.codemag.com/framework	41
CODE Magazine	www.codemag.com/magazine	45
CODE Staffing	www.codemag.com/staffing	61
DevTeach Developers Conference	www.devteach.com	13
dtSearch	www.dtSearch.com	47
LEAD Technologies	www.leadtools.com	5
Tower48	www.tower48.com	75
Xamalot	www.xamalot.com	71

iMessages, F# Type Providers, Swagger, Cordova
CODE
MAR APR 2017
Programming Alexa Skills for the Amazon Echo
Xamarin versus Cordova
Azure Skyline and Docker
Accessing Data with F# Type Providers

Advertising Sales:
Tammy Ferguson
832-717-4445 ext 026
tammy@codemag.com

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

Listing 3: Create a config file to hold the test data source information

```
<configSections>
  <section name="microsoft.visualstudio.testtools"
           type="Microsoft.VisualStudio.TestTools.
                 UnitTesting.TestConfigurationSection,
                 Microsoft.VisualStudio.
                 QualityTools.UnitTestFramework"/>
</configSections>

<connectionStrings>
  <add name="Sandbox"
      connectionString="Server=Localhost;
                        Database=Sandbox;
                        Integrated Security=SSPI"
      providerName="System.Data.SqlClient" />
</connectionStrings>

<microsoft.visualstudio.testtools>
  <dataSources>
    <add name="Sandbox"
        connectionString="Sandbox"
        dataTableName="tests.FileProcessTest"
        dataAccessMethod="Sequential"/>
  </dataSources>
</microsoft.visualstudio.testtools>
```

Sample Code

You can download the sample code for this article by visiting my website at http://www.pdsa.com/downloads. Select PDSA Articles, and then select "Code Magazine—Data-Driven Testing" from the drop-down list.

```
returnValue = Convert.ToBoolean(
   TestContext.DataRow["ReturnValue"]);
causesException = Convert.ToBoolean(
   TestContext.DataRow["CausesException"]);
```

After retrieving the values, you now call the **Assert.AreEqual()** method to compare the return value for the current row with the value returned from calling the FileExists() method, passing in the file name retrieved from the current row. If an exception occurs, it could be caused by two reasons. The first is that the **Assert.AreEqual** method raises an exception of the type AssertFailedException to inform the unit testing framework that this test failed. The second reason is that the FileExists method raises an exception in response to a bad file name being passed in. If this latter situation is the case, you need to check the value you retrieved from the column CausesException in the current row. If it's a true value, the test succeeded; otherwise, the Assert.IsTrue causes an exception of the type AssertFailedException, which again, informs the unit test framework that the test failed.

Move the Connection String to a Config File

Just like any other class, a test class like FileProcessTest should follow the same best practices you would normally employ for your application code. In the DataSource attribute in **Listing 2**, there's a hard-coded connection string and table name. This information is best put into a configuration file to make it easy to change. Microsoft gave us the flexibility to accomplish this. Create an App.config file in the test project and add the items within the <configuration> section shown in **Listing 3**. NOTE: The type= attribute should all be on one line. It is broken into multiple lines to format for this article.

Once this configuration file has been created, modify the DataSource on your FileExistsTestFromDB method to look like that shown in the code snippet below. The DataSource attribute is now a simple reference to the name attribute within the <dataSources> collection. You may have as many different data sources as needed by your tests.

```
[TestMethod()]
[DataSource("Sandbox")]
public void FileExistsTestFromDB()
{
  ...
}
```

You should now be able to run your three unit tests from the information contained in your data table. Comment out the original tests you wrote in this article so the only test method left is the one with the DataSource attribute. Run the tests to ensure that you have everything configured correctly.

Architect Your Code for Testing

Correctly architecting an application will do wonders for re-usability, readability, upgradeability, maintainability, and testability. Take advantage of the design patterns that exist today, such as MVC, MVP, MVVM, or derivatives thereof. All of these patterns help you remove code from the user interface layer of your application. Removing code from the UI layer is the single best thing you can do to have a well architected, easily testable application.

Removing code from the UI layer is the single best thing you can do to have a well architected, easily testable application.

Your UI code should strive to just call or bind to properties and methods in classes. Each class you write should contain all of the logic for your application. By moving all application logic out of the UI and into classes, you make it easy to use the various unit testing tools in Visual Studio. Furthermore, these classes should be combined into assemblies to aid not only in testing, but also in re-usability.

Summary

If you find that you're writing many different unit tests to test a single method that has a lot of inputs and outputs, the data-driven approach outlined in this article may be just what you are looking for. I recommend using a separate database for all your test tables. I also like creating a schema with the name of "tests," as I did in this article. Be sure to put all of your connection strings and data source names in an App.config file for the ultimate in flexibility. Good luck with your testing!

Paul D. Sheriff

CODE

DEVTEACH

MONTREAL JULY 3-7 2017

THE WEB DEVELOPMENT CONFERENCE

DevTeach is going ***Open Source*** all the way !
Our next event - *Open DevTeach Montreal 2017* - will be taking place from July 3rd to July 7th, 2017 at the Delta Centre-ville. Focus will be on Web Open Source Innovations.

Presented during the Montreal 375th anniversary, the Montreal Jazz Festival and the Montreal Fireworks Festival, **Open Devteach Montreal** is the place to be in 2017 to celebrate and learn about Web Open Source Innovations.

Pre-Conference Workshops, Monday July 3rd, 2017

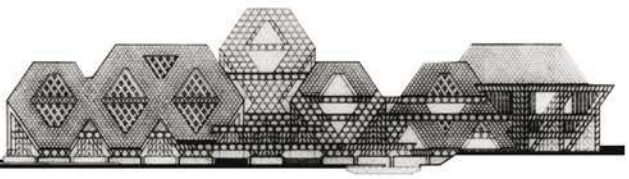
A Day of Securing ASP.NET Core Applications and APIs
Brock Allen

Post-Conference Workshops, Friday July 7th, 2017

Angular 2 Workshop: From Zero To Hero Jr in 1 day !
Laurent Duveau

www.devteach.com
Sponsored by

CODE MAGAZINE ANGULAR ACADEMY



Montreal 375th Anniversary Expo 67 50th Anniversary

Xamarin versus Cordova

Fifteen years ago, launching a product meant writing a Windows application. Ten years ago, it meant writing a website that worked on Internet Explorer. Today, launching a successful product means targeting Mac and Windows, and iOS and Android. And it may also mean targeting tvOS, WatchOS, Tizen, XBOX, Windows Phone, maybe more. It's mind boggling how much times have changed. In previous articles, I've made the case for looking at Angular2 and Typescript, along with Cordova and Electron as a strategy to lower the learning curve and associated costs in targeting multiple platforms. Yes! It's true: write once, run everywhere is finally a reality. You can't target all relevant platforms with a single codebase, and the application looks good, looks consistent, and performs quite well.

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.

But a few years ago, targeting multiple platforms effectively meant that you had to write code for each platform separately. In other words, you wrote "native" code, and didn't use Web-based technologies. The argument in favor of native is that it provides better performance and better access to leading-edge features on the respective platforms.

But writing native has two huge disadvantages: a much higher learning curve and associated costs.

What if you could have the best of both worlds? Write in C# and run everywhere as native applications. That's the value of Xamarin. Xamarin, now part of Microsoft, is an incredible alternative that lets you use one language and one code base and target multiple platforms. So why would you not just use Xamarin, always?

Always! You'll never hear a good developer use that word, although I sometimes run into naysayers. They insist that if you aren't doing 100% native, you're doing it wrong. There's an almost religious war between these two camps. The reality is that good developers take advantage of both platforms where suitable.

In this article, I intend to make the case that it is possible to build your mobile app using the best of each technology. Although Xamarin is great, it has its foibles. Although Cordova is great, it too has foibles. Sometimes mixing the two together gives you the best possible results. I intend to make this case with a sample application written in both Cordova and Xamarin. This sample application has one simple task: to send substantial amounts of data back to the server.

Strengths and Weaknesses

This is perhaps the hardest section to write in this article; I'm presenting strengths and weaknesses on a topic fraught with religiously held opinions and all without proof. Please humor me, as this section is the opinion of one person. The proof is coming shortly.

Vendor Native

By vendor native, I mean Swift or ObjectiveC for iOS and MacOS, Java for Android, and C# and XAML for Windows. The advantage here is best-of-breed performance, the smallest binaries, and the best support from the associated vendor. The disadvantage is having to learn three different platforms, maintain three code bases, and fix the same bug three times, with bonus bugs on top. This approach is expensive.

Xamarin

Using Xamarin means that you write your code in C# but it compiles down to native. The disadvantage is that you'll have a more complex project structure, a more complex dev environment—especially when you need to target iOS from Windows—and performance, though good, is not as good as pure native. Binary sizes are decent, but they're usually larger than pure native.

Also, although it's not technically a disadvantage, it's a fallacy to assume that you won't have to learn the platform intricacies just because you are using C#. The reality is that you still need to understand platform-specific details such as keychain stores, app bundle identifiers, provisioning profiles, and a lot more that comes with any associated platform. Frankly, sometimes the only difference between writing an iOS app using Xamarin and C# and using XCode and ObjectiveC feels like the language. In fact, for 90% of the time I'm stuck on a Xamarin issue, I run a search on the Internet and find an ObjectiveC solution; my workflow is a bit like Google -> copy/paste -> translate ObjectiveC to C#. The language, for the most part, feels like the only major difference. Mind you, that's not a small difference! ObjectiveC is a language only a mother could love. (Swift is not so bad.)

Xamarin Forms

This section will probably get me in the most trouble, but it deserves to be called out. Xamarin forms, in my opinion, have some major disadvantages. XAML (and for that matter even vendor-native controls) are okay at building a UI quickly. However, a significantly branded app is almost never easy to build using XAML.

I realize that there are some XAML experts who've already pulled out their pitchforks. But when it comes to pure developer productivity when comparing HTML5/CSS3 versus XAML, HTML wins hands down. I agree that XAML gives you more fine-tuned control, but HTML adapts better to "reflow" and multiple form factors. The whole issue I'm trying to solve here is maximum reach with minimum effort and cost. On that front, writing up an HTML-based UI is almost always a better choice than trying to do the same thing in XAML or dragging-and-dropping a button and then trying to make that button style completely not like a button. Nobody wants a boring UI, and it might take some real effort to make this button interesting.

The other disadvantage of Xamarin forms is bloat. This productivity abstraction layer creates larger binaries, which, in turn, can affect performance and load times. There are some solutions to this in the linking process, however.

Finally, Xamarin forms is newer than Xamarin itself, so a lot of things you want in a full UX control set are missing. This is a problem that I'm sure will get fixed in due time.

I'm not saying that Xamarin forms are never suitable. If you want a user interface that can be built nicely with drag-and-drop controls with some basic positioning, that doesn't target too many form factors, and isn't heavily branded, it might be okay! In enterprise dev scenarios, sometimes creating numerous functional apps quickly matters more than creating the best-looking apps. But for the typical mobile app that needs ooohs and aaahs, I find that Xamarin Forms are not the easiest route to success, at least not today.

Cordova and Web technologies

The biggest advantage of using Cordova is that these skills are the easiest to find and learn. Also, good-looking user interfaces built using HTML-based technologies can run everywhere, and can be built by people who aren't experts at XCode, Storyboards, or XAML. You can split the problem into smaller manageable chunks and cut your work according to your people, not the other way around. Also, with Angular and Typescript, it's possible, in a practical manner, to write code for mobile and have it run on desktop, and vice versa.

The biggest disadvantage here is performance. But let's dissect this further. Performance, although important, isn't everything. Rather, let's put it this way: your team's performance is important too. For instance, let's talk about HTML's biggest advantage: a cross-platform, responsive user interface that's attractive and easy to build.

It's true that pressing an HTML div that's disguised as a button may take five milliseconds more than, say, a pure native button. But the important take-away here is that the user's not going to care about or even notice five milliseconds. Of course, if you did a button press 100,000 times in a loop, those five milliseconds add up to something substantial, and then the user will notice and care.

JavaScript engines have gotten pretty good. Your regular business logic, such as those that enable a textbox when the checkbox is checked, etc., run pretty much as fast as native. Where you pay a huge penalty is when you start marshalling between JavaScript to native and vice versa. Especially when you start marshalling in a loop.

That's the number one target you need to address: where you marshal across JavaScript to a native boundary repeatedly over thousands of times, a Cordova operation could amount to delays measurable in seconds, if not worse. This is something that the user will notice.

The other situation you need to consider are specific UI elements, such as scrolling list views with large amounts of data, or map applications with pan and zoom elements that have very large images; the sorts of things that could be done in HTML but are done better in native. The good news is that Cordova applications let you replace one screen with native implementation, one UI element with native implementation, or they simply ditch the whole Cordova shell and still use the same concept to write hybrid apps in either vendor native, or using Xamarin.

The Important Take-Away

The important take-away here is that every technology has its strengths and weaknesses. And the mobile application development landscape is changing rapidly, so what's true today in this article may change a year from now. I feel that crafting up your UX in HTML is a very viable approach, as long as you know when to drop HTML and JavaScript and dive into native code.

You've probably heard that dropping into Native code is expensive, and that you have to write it three times in three different platforms. It's not the case! This is where Xamarin comes into the picture. You write your code once, as a combination of C#/Xamarin, and HTML, CSS, and JavaScript.

With judicious mix-and-match, you now have the performance of native and the flexibility of HTML, all with a single code base. Simply amazing!

Listing 1: A very simple Web API

```
public class TestController : ApiController
{
    public void SubmitData(string inputData)
    {
        System.Diagnostics.Debug.WriteLine(inputData);
    }
}
```

Listing 2: The sendData function

```
sendData:function() {
    var now = window.performance.now();
    var deferreds = [];
    var i = 1;
    for (i = 1; i <= 10000; i++) {
        deferreds.push(sendDataChunk());
    }
    var then = window.performance.now();
    document.getElementById(
     "sendDataButtonLog").innerHTML =
     "Total time (ms): " + (then - now);

    $.when.apply($, deferreds).done(function() {
        // you can check for total time
        // for request to complete here.
    });
    function sendDataChunk() {
        var deferred = new $.Deferred();
            var postUrl = "..removed..";
            $.ajax({
                type: "POST",
                url: postUrl,
                success: function() {
                    deferred.resolve();
                }
            });
        return deferred;
    }
}
```

Listing 3: The HybridWebView view.

```
using System;
using Xamarin.Forms;

namespace HybridView
{
    public class HybridWebView : View
    {
        Action<string> action;
        public static readonly BindableProperty UriProperty =
        BindableProperty.Create(
            propertyName: "Uri",
            returnType: typeof(string),
            declaringType: typeof(HybridWebView),
            defaultValue: default(string));

        public string Uri
        {
            get { return (string)GetValue(UriProperty);}
              set { SetValue(UriProperty, value) ;}
        }

        public void RegisterAction(Action<string> callback)
        {
            action = callback;
        }

        public void Cleanup()
        {
            action = null;
        }

        public void InvokeAction(string data)
        {
            if (action == null || data == null)
            {
                return;
            }
            action.Invoke(data);
        }
    }
}
```

Listing 4: The HybridViewPage.xaml file

```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage
       xmlns="http://xamarin.com/schemas/2014/forms"
       xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
      xmlns:local="clr-namespace:HybridView"
       x:Class="HybridView.HybridViewPage">
    <ContentPage.Content>
       <local:HybridWebView x:Name="hybridWebView" Uri="index.html"
                HorizontalOptions="FillAndExpand"
                VerticalOptions="FillAndExpand" />
    </ContentPage.Content>
</ContentPage>
```

The App

I'm going to illustrate my argument here by building an app that posts data to a Web API written in ASP.NET. To keep things germane, I'm keeping the WebAPI extremely simple. There's no authentication and there's no complex payload. It's just a simple action that accepts a string on HTTP POST. This reflects a real-world scenario, where the app has a large payload to post. The app isn't going to post that entire payload in one request. This not only has memory implications on the mobile device, but the request itself may time out. You want to break apart that request into smaller portions. The Web API can be seen in **Listing 1**.

This API can be called on a URL like this:

```
/api/test/SubmitData?inputData="sampleinput"
```

Aren't plug-ins how you're supposed to target native capabilities in Cordova?

With the API out of the way, let's write the apps. The app does a very simple task: It calls the API 10,000 times. You'll measure the time it took to send those requests; note that you're not interested in measuring how long it actually took for the server to respond. You're only interested in the operation of firing these requests. The reason you want to measure the time it took only to send the requests is so that you can get a decent comparison of native versus JavaScript, and see if there's any merit to all the fuss some people are making regarding native performance being better. You don't want network traffic or other such extraneous factors to affect this measurement.

The Cordova App

I'm going to skip over Cordova basics here and just talk of the main steps I took to create the app.

I first created a basic Cordova project and then added jQuery into it. I cleaned up the user interface to remove that device-ready message that the starter Cordova app shows, and instead added a button and a div, as shown in this snippet:

```
<div id="deviceready">
    <br/>
    <button id="sendDataButton">
      Send Lots of Data</button>
    <div id="sendDataButtonLog"></div>
</div>
```

Also, because I intend to make HTTP calls from the application, I added the following to the "Content-Security-Policy" meta tag:

```
;connect-src http://*/*
```

Note that the above is very insecure: it allows calls to any HTTP URL. It's okay for testing purposes.

Next, let's modify the JavaScript portion of the code. The idea is that when the **sendDataButton** is clicked, you wish to fire off 10,000 POST requests and measure how long it took to send those requests. You don't care about how long it takes for those requests to return.

The first thing to do here is to add an event handler for the **sendDataButton** click event. You can do this in the initialize function, as shown in this next snippet:

```
$("#sendDataButton").bind('click', app.sendData);
```

As you can see, clicking the button calls a function called sendData on the app object. This function can be seen in **Listing 2**.

As **Listing 2** shows, you're firing off 10,000 requests, but you don't wait for them to finish. You're simply interested in measuring the time it took to fire off the requests. In doing so, you're going from JavaScript to native, although all of this is implemented in the WebBrowser itself. You can imagine that going to the Cordova shell would probably be an even more expensive operation. An example of going to the Cordova shell might be calling the Camera API, or perhaps some offline data store API other than window.localStorage.

Now, go ahead and run the application. It shows you a user interface with a button that says "Send Lots of Data." Press that button, and the application informs you of how long it took to send 10,000 requests.

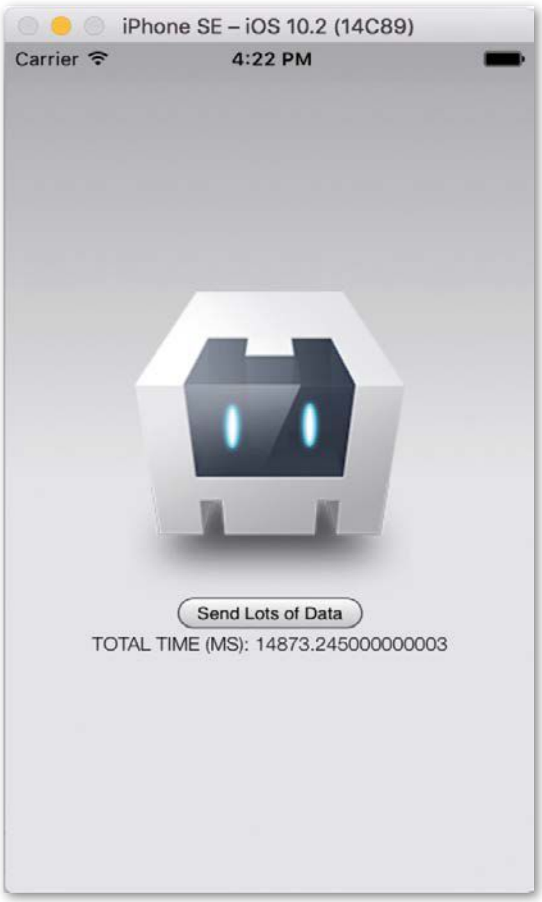
10,000 requests are quite a bit, so, as can be seen from **Figure 1**, it took nearly 14 seconds to send these requests. Now let's repeat this example using native code written in Xamarin.

The Xamarin Hybrid App

Instead of writing a full Xamarin forms-based app, I instead choose to write a hybrid app. A hybrid app, in this case, is a Xamarin app that uses Web-based technologies for its user interface and basic business logic. It has the ability to ask Xamarin native code to do heavy lifting wherever necessary. For instance, here I ask the native code to make the POST requests for us. This is a very good approach because now I can move back and forth between Cordova apps and custom Xamarin Hybrid apps.

What about plug-ins? Aren't plug-ins how you're supposed to target native capabilities in Cordova? Indeed, plug-ins are one way to the goal, but plug-ins usually still mean that you have to write the plug-in in multiple platform-specific languages. This brings me back to my original problem of avoiding multiple code-bases. I could write a plug-in using Xamarin, but in its current incarnation, Xamarin likes to own the entire project. The project structure is not the most conducive to writing a Cordova plug-in. Or I could rely on the community to provide a plug-in for me. But those plug-ins are written for the most generic circumstances.

For instance, imagine that my requirement was to search an offline store using a key. I could use a plug-in to read the entire offline store, and then run it in a loop, going over the JavaScript/native firewall over and over again, and pay-



TOTAL TIME (MS): 14873.245000000003

Figure 1: The Cordova app took some time to send the data.

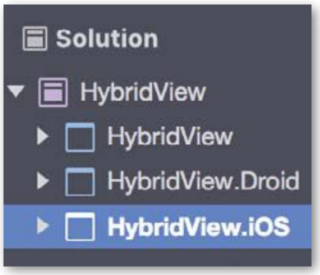


Figure 2: The Xamarin app project structure

Listing 5: HybridViewPage.xaml.cs

```
using System.Net;
using Xamarin.Forms;

namespace HybridView
{
    public partial class HybridViewPage : ContentPage
    {
        public HybridViewPage()
        {
            InitializeComponent();
            hybridWebView.RegisterAction(data => sendData());
        }

        private void sendData()
        {
            var now = System.DateTime.Now;
            string url = "..removed..";
            for (int i = 0; i < 10000; i++)
            {
                HttpWebRequest request =
                    (HttpWebRequest)WebRequest.Create(url);
                request.Method = "POST";
                request.BeginGetResponse(
                    (ar) =>
                    System.Diagnostics.Debug.WriteLine(
                        "end:" + ar.AsyncState.ToString()), i);
            }
            var then = System.DateTime.Now;
            DisplayAlert("Alert",
                         "Total time (ms):" +
                         (then - now).TotalMilliseconds,
                         "OK");
        }
    }
}
```

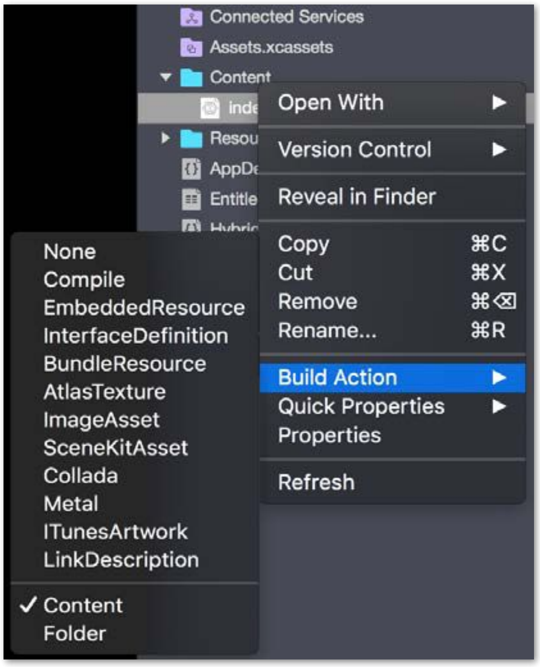


Figure 3: Build action for the index.html file

ing a big performance penalty. Although this is a contrived example, and for such common scenarios, plug-ins exist that let you pass in the **Search** key once. Because the iteration is done purely in native code, it's quite conceivable that a plug-in matching your specific business requirement doesn't exist.

You could write it using ObjectiveC, Java, WinJS, and God knows what else! Or, you could avoid that complexity all together and use Xamarin for native, and HTML and JavaScript for UI and basic business logic. At least, that's what I like to do, because it seems to be the path to success that requires the least effort.

Let's get started.

The first thing to do is to create a Xamarin project. Start Visual Studio for Mac (if you have a Mac), or just Visual Studio with Xamarin tools installed, and choose to create a **Forms App** project using C#. Ensure that the shared code project is a portable class library, and choose to use XAML for user interface files. Call this project **HybridView**.

Once the project is created, you should see a project structure like that shown in **Figure 2**.

Next, in the HybridView project, add a new class called **HybridWebView**, and put the code shown in **Listing 3** into it.

As you can see from **Listing 3**, you're creating a simple view that accepts a URI as a bindable property. Also, it allows you to invoke an action. The idea is that you can specify what page to load using the bindable property, and that page can invoke native code functionality using the action.

Next, use this view in the HybridViewPage ContentPage. Go ahead and edit its XAML portion, as shown in **Listing 4**.

As can be seen from **Listing 4**, I'm using the View I created and I'm pointing it to a file called **index.html.** I'll create this file in the HybridView.iOS/Droid projects and embed it as content. You can also load a file over HTTP, but I doubt that Apple would be too pleased with such an app and you run a high risk of app rejection.

Next, you need to handle the action that can be invoked by the JavaScript code, and in native code, fire off 10,000 POST requests and measure the time it took to send those requests. As before, you won't wait for the results to return. This code goes in the HybridViewPage.xaml.cs file, and can be seen in **Listing 5**.

Great! All that's left to be done is to create a simple index.html file that can call this action, and a view renderer that can load this HybridWebView view. Let's first create the view renderer as a class in the HybridView.iOS project, which can be seen in **Listing 6**.

For the index.html, create a folder called **Content** in the HybridView.iOS project and drop an index.html in that folder. Make sure that its build action is set to Content, as can be seen in **Figure 3**.

All that's left now is to put some simple code in the HTML file that calls the native function, which can be seen in the next snippet:

```
<button type="button"
    onclick="invokeNativeCode('hello');">
```

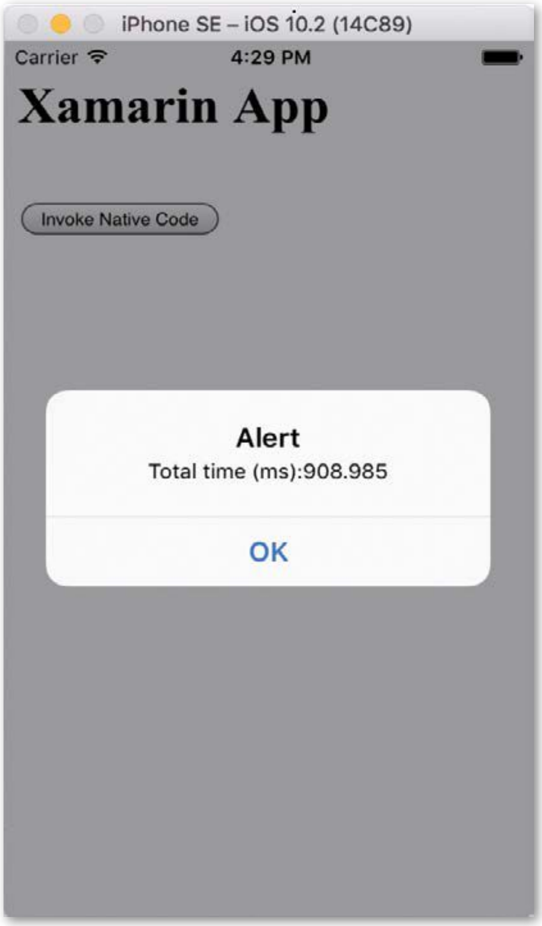


Figure 4: The Xamarin app doing the same job as the Cordova app

Listing 6: The HybridWebViewRender

```
using System.IO;
using HybridView;
using HybridView.iOS;
using Foundation;
using WebKit;
using Xamarin.Forms;
using Xamarin.Forms.Platform.iOS;

[assembly: ExportRenderer(typeof(HybridWebView),
typeof(HybridWebViewRenderer))]
namespace HybridView.iOS
{
  public class HybridWebViewRenderer :
  ViewRenderer<HybridWebView, WKWebView>, IWKScriptMessageHandler
  {
    const string JavaScriptFunction =
    "function invokeNativeCode(data){
    window.webkit.messageHandlers.invokeAction.postMessage(data);}";
    WKUserContentController userController;

    protected override void OnElementChanged(
        ElementChangedEventArgs<HybridWebView> e)
    {
      base.OnElementChanged(e);

      if (Control == null)
      {
        userController = new WKUserContentController();
        var script = new WKUserScript(
        new NSString(JavaScriptFunction),
         WKUserScriptInjectionTime.AtDocumentEnd, false);
         userController.AddUserScript(script);
        userController.AddScriptMessageHandler(this,
          "invokeAction");

        var config = new WKWebViewConfiguration {
            UserContentController = userController };
        var webView = new WKWebView(Frame, config);
        SetNativeControl(webView);
      }
      if (e.OldElement != null)
      {
        userController.RemoveAllUserScripts();
        userController.RemoveScriptMessageHandler("invokeAction");
        var hybridWebView = e.OldElement as HybridWebView;
        hybridWebView.Cleanup();
      }
      if (e.NewElement != null)
      {
        string fileName =
         Path.Combine(NSBundle.MainBundle.BundlePath,
           string.Format("Content/{0}", Element.Uri));
           Control.LoadRequest(
             new NSUrlRequest(new NSUrl(fileName, false)));
      }
    }

    public void DidReceiveScriptMessage(
     WKUserContentController userContentController,
     WKScriptMessage message)
    {
      Element.InvokeAction(message.Body.ToString());
    }
  }
}
```

```
    Invoke Native Code
</button>
```

Go ahead and build this app and run it in the same simulator or device that you ran the Cordova project in. The user interface shows you a button that you can click on. Go ahead and click on it. Clicking on the button fires off 10,000 POST requests, and shows you the time taken for those requests to be sent. This can be seen in **Figure 4**.

As is quite clear, the Xamarin app, being native, is orders of magnitude faster than the Cordova app. Subsequent executions show that the Xamarin app for this specific task is four to 15 times faster. That is quite a bit—it's performance that the user will notice.

But let me qualify that; you're still using an HTML-based user interface. I'm sure you're paying a slight performance penalty for it, but, the user doesn't notice the extra five milliseconds it took to click an HTML button versus a native button. In fact, in many instances, an HTML UI may perform *better* than a native UI.

But the main point here is that being able to mix and match when creating a user interface in Web technologies gives you a lot of flexibility in crafting up a responsive, reflowable user interface using a single portable skillset. Where you need to dive into native, you do so using Xamarin. This gives you the best of all the worlds.

Although I'm sure that going vendor native will perform even better, the effort to do so would be three times, if not more, and the improvement with vendor native, frankly, isn't that much beyond what Xamarin native already gives you. That isn't to say that I never find myself opening XCode.

Summary

In a land far, far away, there's a battle brewing between developers. Some insist that native is the only way you should write apps. Others insist that writing apps using native offers very little benefit and delivering Cordova apps is the right way to serve your user's needs.

Thankfully, I live far, far away from that land of battles. I feel that the mobile dev landscape is changing very rapidly, and it beckons you to stay on top of the changes so you know what's the best way to build an app today. From my point of view, right now, writing user interfaces in HTML is the easiest. That native code offers better performance seems to hold true, but those situations where you must confine your efforts to native code are rare. I also know that Xamarin offers 90% of the performance benefit of vendor native, while also giving me a portable codebase that I can share across platforms.

Thankfully, it's possible for me to pick the right tool for the right job. Sometimes it's Xamarin, sometimes it's Cordova, and sometimes it's vendor native.

I hope you found this article useful. Until next time, happy coding!

Sahil Malik

Legal Notes: Selecting and Operating a Legal Entity for Your User Group

Most user groups start off as a loosely formed confederation of like-minded developers who want to build a learning community. Over time, as a user group grows, it may want to start hosting events, an activity that often necessitates the need to enter into agreements for venue space and to solicit funds. When a user group decides to take this next step, the user group can no longer operate as it once did. Rather, it must organize as a legal entity so that it can, among other things, open a bank account. This is the most often-cited reason why user groups organize as a formal business entity.

John V. Petersen, Esq.
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. For over 20 years, John has developed, architected, and designed software for a variety of companies and industries. John is also a video course author for Lynda.com. 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.

This article is for two basic classes of people. The first is the group of people that either have a user group or want to form a user group and want to take it to the next level via a business entity. The second is the group of people that are part of a user group that is organized under a formal business entity but aren't sure how to properly run such an organization.

DISCLAIMER: This and future 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.

Types of Business Organizations

There are a handful of business organization types that your jurisdiction likely recognizes and these are enumerated below. The most common type of organization for a user group to select is that of a Non-Stock Non-Profit Corporation. The other organization types are listed for informational and differentiation purposes.

- **Sole Proprietorship:** As the name implies, this is a single individual where the individual registers a fictitious name with his jurisdiction for the purposes of obtaining a Tax ID Number (TIN). Note: sole proprietorships are not distinct business entities like Partnerships, Corporations, and Limited Liability Corporations (LLCs).
- **Partnerships**: Partnerships are business entities where two or more individuals and/or entities enter into an agreement and register under a given name with their jurisdiction. There are several different types of partnerships including General Partnerships (GP), Limited Partnership (LP), and Limited Liability Partnerships.
- **Corporations:** Corporation types can be broken down into two main types: profit and non-profit. For a user group, the most applicable type is that of non-profit for the simple reason that the purpose of the organization is not to make a profit to distribute to shareholders. Rather, a user group non-profit is usually organized for an educational mission. Typically, non-profits are organized on a non-stock basis, meaning that unlike a for-profit corporation that is owned by shareholders, a non-profit is typically in the care of and run by a board of directors for the benefit of its members or some other community. Corporations, unlike partnerships and sole proprietorships, are distinct entities unto themselves and stand apart from any individual person.
- **Limited Liability Corporations (LLCs):** LLCs are a hybrid of the partnership and corporate model. LLCs retain the simplicity of a partnership while maintaining the liability shield of a corporation.

Just because there aren't profits involved, don't think for a moment that it's all fun and games.

Because a user group is typically organized for non-profit purposes, the only feasible entity is the non-profit corporation structure. Just because there aren't profits involved, don't think for a moment that it's all fun and games. In most jurisdictions, the most heavily regulated and scrutinized businesses are non-profits. The reason is that because most non-profits run correctly under the law, non-profits are often used as part of fraudulent and criminal enterprises. As it turns out, running a non-profit is not that difficult. Like anything, you just need to understand and respect the rules.

Creating Your Non-Profit

Creating a corporation is, to many, surprisingly simple. You only need a few things to get started:

- A unique name
- Articles for incorporation, which is a document that sets forth the basic mission of your organization, its name, and address. Depending on your jurisdiction, the articles may also indicate whether your organization has members or not and whether the non-profit is organized on a stock or non-stock basis.
- An initial slate of officers/board members (president, secretary, and treasurer, at a minimum)
- The filing fee

You may have heard about a document known as by-laws and may be wondering why I didn't mention them above. The reason is that in spite of their importance, by-laws are not required to form a non-profit corporation. By-laws set forth the parameters of what the board of directors are allowed to do on behalf of the corporation. By-laws also specify things such as the maximum and minimum number of board members, when and how elections occur, how board vacancies are filled, etc. In the absence of specific by-laws, most jurisdictions codify a number of default rules to govern board of director conduct.

Once you file your paperwork, you now have a corporation and you can then obtain a Tax ID Number (TIN) from the Internal Revenue Service (IRS). Once you have a TIN, you can open a bank account.

Observing Corporate Formalities Isn't Optional!

One of the things that sets the corporation structure apart from other structures is the need to observe what is collectively known as corporate formalities. This includes regular business meetings where minutes are taken and votes recorded for actions the board takes. In the case of a user group, having a user group meeting may satisfy this requirement.

A good user group meeting is divided into two parts. The first is the business part where a meeting is called to order to discuss the organization's business. This often includes an approval of minutes from the previous meeting, a financial report, committee reports, etc. The second part of the meeting is where the fun stuff happens and you talk about whatever it is that brought you all together.

It's important to understand that a corporation is a distinct entity and it must be run as such. Otherwise, the organization ceases to operate as a bona-fide entity and instead becomes an alter ego for one or a few people. This is when there can be a great amount of unintended consequences and legal trouble.

Annual Meeting

At the very least, a corporate entity must have an annual meeting to, among other things, hold elections for the board and offices such as President, Vice-President, etc. Annual meetings also afford an opportunity to ratify actions taken in the current year, approve budgets, and ensure legal compliance. Well-run non-profits hold business meetings several times a year, whether they are monthly, quarterly, or semi-annually.

Annual Filing Requirements

Depending on what your user group non-profit does and the requirements of your jurisdiction, you may have annual filing requirements to satisfy.

By-Law Amendments

As time goes on, the organization may change and the by-laws may have to change as well. There is a formal process to amend by-laws that is typically stated in the by-laws themselves.

If You're Not Observing Corporate Formalities, What's the Big Deal?

If your group is dormant and doesn't have any activities, most likely, there's no big deal. In that case, the group should consider formal dissolution procedures. If, on the other hand, yours is an active group that, among other things, solicits, receives, and disburses money, the consequences can be severe.

Remember that a corporation is a distinct entity and that a board of directors acts on behalf of the corporate entity. Let's assume for a moment that a corporation with members has been in existence for 10 years, but has never held an election nor had any business meetings and that votes were never taken. Assume further that during this time, funds were solicited and spent in the *name of the corporation*. In such a case, there's a strong argument to conclude that such funds where fraudulently solicited and spent because there was no authorization to do so. Again, it's important to stress that a corporation is a distinct entity and that it's the rules of the entity that confer the authorization to act on its behalf. In this particular case, there are problems with the solicitation of funds and the withdrawal of funds from the organization's bank account. Depending on your jurisdiction, the amounts involved, and other circumstances, criminal conduct may be involved that can be graded as a misdemeanor or a felony.

Director Liability and Avoiding Conflict of Interest

Ultimately, the caretaking and well-being of an organization depends on its board of directors. The idea is that although the board and members come and go, the organization will endure. This is why for non-profits; board members typically serve on a volunteer basis. If there's any malfeasance on the part of the organization, the directors, both as a group and individually, are on the hook legally.

What is allowed? That depends on what the by-laws state. In the absence of by-laws, what a board and organization is allowed to do is narrowly construed to be just enough to support the mission outlined in the articles of incorporation.

The watershed moment for most groups is when they start dealing with money and further, when they pay out money to individuals. This is where non-profits have to be very careful. As a rule, to undertake such actions, there must be by-laws stating that such actions are permissible. One issue that's often raised is if a board member can be compensated. As a rule, the answer is a hard and fast no. The only way around that is to have a provision in the by-laws that carves out an exception for directors. If such an exception does not exist, the by-laws can be amended in accordance with the procedures set forth in your organization's by-laws or your jurisdiction.

Conclusion

If you find yourself in need of forming a business entity for your user group, you need to understand that it won't be business as usual anymore. There are a number of regulations, for good reason, that govern how non-profits are operated. When dealing with money, extra care must be taken to avoid any actual or appearance of impropriety. As a best practice, have a well defined set of by-laws that clearly outlines the actions the organization and directors are allowed to undertake.

If there's one word that you'll want to keep in mind, it's the word transparency. Always be transparent in your dealings with open meetings. Most importantly, be exceedingly transparent with your finances. At the end of the day, you want your organization to transcend any one person or group of individuals so that although individual leadership changes, the organization will endure.

John V. Petersen

CODE

"Non-Profit" Guidance

A good resource for those wishing to have guidance on how to operate a non-profit is the National Council of Non-Profits: https://www.councilofnonprofits.org/tools-resources/principles-and-practices. I've also written extensively about a particular user group's difficulties with proper non-profit governance here: https://www.linkedin.com/pulse/another-example-how-user-group-organized-non-profit-entity-petersen.

Programming Alexa Skills for the Amazon Echo

The Amazon Echo devices are incredibly useful products that come with a sparkling personality (Alexa) and a huge library of skills, with more being added every day. But there's a lot more to them than just playing songs and telling jokes.In this article, you'll learn how Alexa works, and how to develop, deploy, and test your own Alexa Skill in C#, and how to deploy it to Amazon for everyone to use.

Chris G. Williams
chrisgwilliams@gmail.com
www.geekswithblogs.net/cwilliams
twitter.com/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's 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.

Introducing Alexa

You may see the terms Alexa and Echo tossed around interchangeably online, so to avoid any confusion, it's important to get some basic terminology in common (see **Table 1**) before I begin.

Using Alexa

It all starts with you, the user, asking/telling Alexa to do something (via the Amazon Echo device). From there, Alexa takes your request and converts it to text. The text command is parsed to identify the skill being requested, and then routed to the proper skill service endpoint for processing. Then, the response is sent back from the service to Alexa, converted back into speech, and finally played for the user to hear.

If you've never used Alexa before, **Figure 1** illustrates the User Interaction Flow.

In practice, the command is structured like this: **Wake Word + Phrase + Invocation Name + Intent (+ optional limit)**

Examples:

- Alexa {verb} {SkillName} to {Do Something}
  - Alexa ASK WheresMyBeer for the three nearest breweries.
  - Alexa TELL WheresMyBeer to get me a list of breweries near me.
  - Alexa PLAY the newest Norah Jones album.
  - Alexa TURN OFF the living room lights.
  - Alexa SET AN ALARM for 10 minutes from now.

Other supported commands include: Begin, Launch, Load, Open, Resume, Run, Start, Talk to, and Use.

Example: Alexa RUN SurfReport AND GET ME the high tide.

You can use the following prepositions: About, For, If, To, Whether, and "blank" (to return a response prompting for more information.)

Example: Alexa TELL SurfReport TO GET ME the high tide FOR TOMORROW.

Alexa is great for returning lists of information, but you can also be more restrictive in your requests by using Limits with your Intents. You're provided with a list of built-in Amazon intent types (see the list: http://amzn.to/2aw07gw), which include: AMAZON.DATE, AMAZON.DURATION, AMAZON.FOUR_DIGIT_NUMBER, AMAZON.TIME, AMAZON.NUMBER, and AMAZON.LITERAL. You can also create custom intent limit types, just like in .NET.

Be sure to read the article "*Understanding How Users Invoke Custom Skills,*" located at http://amzn.to/1UmXGjz

Term	Definition
Amazon Echo	This is the hardware product. Currently available in two formats: Echo and Echo Dot: • Activated by a wake word, these are functionally the same, although the Echo Dot has a much smaller speaker. • Amazon Tap and Fire TV Remote: Both are activated by touch and then respond to voice commands.
Alexa	This is the personality of the Amazon Echo device and also the "wake word" you use to get Alexa's attention. (You can also set the wake word to use "Amazon" or "Echo.")
Wake Word	A word that Alexa listens for, to be followed by a command. Other (optionally configurable) wake words include "Amazon" and "Echo." If you have an Android phone or tablet, you're probably familiar with the "OK Google" wake word, and of course, Windows 10 and Windows 10 Phone users have Cortana, who listens for "Hey Cortana."
Alexa Voice Service	This is the cloud-based brain behind the voice, and is also responsible for translating voice to text and back again. Handles all natural language processing and interpretation.
Alexa Skill	This is a function that Alexa knows how to perform. Common Alexa Skills include playing music, playing games, telling jokes, checking your calendar, controlling devices via a smart hub, and a lot more.
Companion App	In order to configure the Amazon Echo, and Alexa, as well as any external hardware and smart home devices, you need to run the companion app (currently available for Android and iPhone). For an enhanced Alexa Skill experience, you can push additional information and graphics to the companion app, in addition to the usual voice response via Alexa.

Table 1: Alexa Terms and Definitions

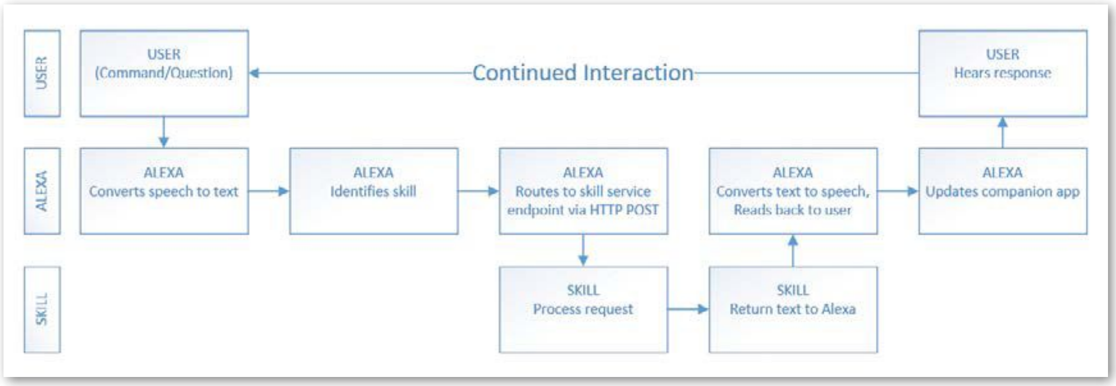

USER
ALEXA
SKILL
USER (Command/Question)
Continued Interaction
USER Hears response
ALEXA Converts speech to text
ALEXA Identifies skill
ALEXA Routes to skill service endpoint via HTTP POST
ALEXA Converts text to speech, Reads back to user
ALEXA Updates companion app
SKILL Process request
SKILL Return text to Alexa

Figure 1: The Alexa User Interaction Flow.

for a much more in-depth discussion of command structures and built-in types.

For more information, read the article "Understanding How Users Invoke Custom Skills," located at http://amzn.to/1UmXGjz.

Starting Your First Alexa Skill

When developing Alexa Skills for the Amazon Echo, you'll be spending a lot of time at the Alexa Developer Portal (http://developer.amazon.com/alexa), so you should take a moment to familiarize yourself with it now.

In addition to some rotating featured content, you'll find some "getting started" links and some "Why Alexa?" information. Near the top, you'll also find links to the following three sections:

- Alexa Voice Service
- Alexa Skills Kit
- Alexa Fund

I'll go over all three sections in this article, starting with the Alexa Voice Service.

Alexa Voice Service

The Alexa Voice Service (AVS) is the cloud-based speech-recognition and natural language processing service for the Amazon Echo/Alexa. Because everything is cloud-based, Amazon can roll out enhancements and updates as needed, and Alexa continues to grow smarter and smarter.

In addition to supporting the Amazon Echo devices, the Alexa Voice Service also allows you to voice-enable any connected product that has a microphone and a speaker, like a smart watch, TV, refrigerator, or even your car.

If you're looking to create commercial-grade physical products that use Amazon Alexa, you can also purchase hardware development kits relatively inexpensively to prototype with.

Alexa Skills Kit

The Alexa Skills Kit, also referred to as the ASK, is Amazon's free Alexa SDK. The ASK comes with a lot of great samples and documentation. To get started, pull down some of the sample code and take a look at how things are set up before jumping into building your own skill in C#.

Amazon likes to move pages around occasionally, so rather than just giving you a link that might break later on, I'm going to guide you down the yellow brick road, so to speak, in order to find the information you seek:

1. Starting at the Amazon Alexa Skills Kit page, click the "*Get Started With Our Step By Step Checklist*" link (http://amzn.to/2cb5a9o).
2. Click the "*Getting Started Guide*" link (http://amzn.to/1UczDnl).
3. Click the "*Using the Alexa Skills Kit Samples*" link (http://amzn.to/2hFt50M).

On the samples page, you'll see a section called "*Getting the Source Code for the Samples and the Java Library,*" which contains links to the JavaScript (https://github.com/amzn/alexa-skills-kit-js) and Java (https://github.com/amzn/alexa-skills-kit-java) samples, along with some additional info about each.

Remember, there are no C# samples yet. We're brave explorers forging a path through uncharted territory! Don't worry though, we're on this adventure together.

If you haven't already, click the link above and grab the **JavaScript** sample. Once you're at the GitHub page for the *Node.js Alexa Skills Kit Samples*, look for the green button on the right that says **Clone or Download**.

If you're familiar with GitHub, and have Git set up locally already, feel free to clone the repo, otherwise you can just download the **alexa-skills-kit-js-master.zip** file to your local computer and extract it. It's not very large, weighing in at roughly 8.5 MB.

There are a lot of great samples to choose from (see **Figure 2**), and I'll be focusing on two of them right now (although I encourage you to look at all of them. Pick them apart, deploy them, and play with them.

Every Alexa Skill sample project contains a speechAssets folder, which, in turn, contains (at least) two files: In-

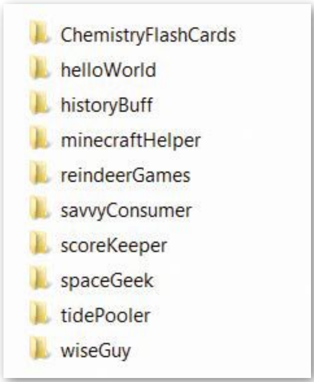

ChemistryFlashCards
helloWorld
historyBuff
minecraftHelper
reindeerGames
savvyConsumer
scoreKeeper
spaceGeek
tidePooler
wiseGuy

Figure 2: The Sample Alexa Skill Projects

tentSchema.json and SampleUtterances.txt. These two files are the heart of defining a good voice interface, which is the key to a robust user experience.

There are no C# samples yet. We're brave explorers forging a path through uncharted territory!

The **HelloWorld** sample is definitely the simplest, so I'll take a look at it, and the **MinecraftHelper** sample (which is relatively complex by comparison) before jumping in to building your own.

The intentSchema.json file is where you define the features (called Intents) of your Amazon Alexa Skill. Unfortunately, the IntentSchema.json for the HelloWorld sample isn't particularly interesting.

Inside the Intents array, there's a HelloWorldIntent and one of the built-in intents (more on that in a minute): AMAZON.HelpIntent.

```
{
  "intents":
  [
    {
      "intent":"HelloWorldIntent"
    },
    {
      "intent":"AMAZON.HelpIntent"
    }
  ]
}
```

The IntentSchema.json for the MinecraftHelper sample is a little more varied, because it accepts an argument (item) and has more of the built-in AMAZON intents included.

```
{
  "intents":
  [
    {
      "intent": "RecipeIntent",
      "slots":
      [
         {
           "name":"Item",
           "type":"LIST_OF_ITEMS"
         }
      ]
    },
    {
      "intent":"AMAZON.HelpIntent"
    },
    {
      "intent":"AMAZON.StopIntent"
    },
    {
      "intent":"AMAZON.CancelIntent"
    }
  ]
}
```

Slots (Amazon lingo for parameters) are how you define the data and type your intent accepts. You can only use them on your own intents though, because the built-in intents don't allow it. In the MinecraftHelper schema, there is one slot, named **Item**, which you'll see again in your SampleUtterances file in just a bit.

The built-in intents are super useful, because you don't have to write any special code to implement them. I'll dig into them in more detail later, but here's a quick run-down of a few of the most commonly used ones:

- AMAZON.CancelIntent (stay in the skill, cancel a transaction or task)
- AMAZON.StopIntent (stop and exit the skill)
- AMAZON.HelpIntent (provide the user with information on how to use the skill)

You get all of these for free. You don't even have to add them to your schema unless you intend to extend them to customize the utterances that trigger them.

They didn't do it in the MinecraftHelper sample, but if you have several intents listed in your schema, it's a best practice to always put the AMAZON.HelpIntent last! This is because Alexa defaults to the last intent in the schema if it can't otherwise find a match. You can use this to provide the user with some helpful suggestions on how to properly use your skill.

Also, Intent and Slot names are case-insensitive and you can't give them the same name (even with different case). Doing so gives you an error when you try to publish the skill.

Always put the HelpIntent last because Alexa defaults to the last intent in the schema if it can't otherwise find a match.

If the IntentSchema.json file is where you define what capabilities your Alexa Skill has, then the SampleUtterances.txt file is where you define how the user can invoke them. Take a look at the SampleUtterances.txt file from the Hello World project sample:

```
HelloWorldIntent say hello
HelloWorldIntent say hello world
HelloWorldIntent hello
HelloWorldIntent say hi
HelloWorldIntent say hi world
HelloWorldIntent hi
HelloWorldIntent how are you
```

As you can see, there are a number of ways to invoke the HelloWorldIntent, ranging from brief ("hi") to slightly less brief ("say hello world"). The AMAZON.HELPIntent isn't included in the HelloWorld SampleUtterances.txt file, because it's a default behavior (it responds to the phrase "HELP"), but if you wanted to extend it, you could add some lines like this:

```
AMAZON.HelpIntent help me
AMAZON.HelpIntent rut row
AMAZON.HelpIntent oh no
```

This is a plain text file containing a dictionary of phrases, organized as key/value pairs that map the intents in your IntentSchema.json file to the sample spoken phrases.

There are a couple of things to keep in mind with this file:

- The KEY (on the left) must match a name in the Intents Schema document.
- The VALUE is the spoken phrase, and may include slots, such as "Limit" (if you defined the intent to accept them.)

The HelloWorld sample still responds to "help," in addition to the other phrases you just added.

By comparison, the SampleUtterances.txt file in the MinecraftHelper sample project is significantly longer (so long, in fact, that I'm not even going to show you all 120+ lines of it here).

```
RecipeIntent how can I build a {Item}
RecipeIntent how can I build an {Item}
RecipeIntent how can I build {Item}
RecipeIntent how can I craft a {Item}
RecipeIntent how can I craft an {Item}
RecipeIntent how can I craft {Item}
RecipeIntent how can I get a {Item}
RecipeIntent how can I get an {Item}
RecipeIntent how can I get {Item}
RecipeIntent how can I make a {Item}
RecipeIntent how can I make an {Item}
RecipeIntent how can I make {Item}
RecipeIntent how can you build a {Item}
RecipeIntent how can you build an {Item}
```

Even if you have multiple intents defined in your IntentSchema, they all go in the same SampleUtterances file. Whitespace is ignored, so feel free to break them up or group them however you like.

One thing to make note of, especially in the last sample, is the use of the "a" and "an" articles, as well as their omission. The more detailed you make this file, the better, because you don't know how the user pronounces these expressions. One of the other samples in the zip file you downloaded includes options like "I don't know" and "dunno" in the list of acceptable utterances.

The Alexa Voice Service (which I'll get to in a bit) is quite good. Depending on your enunciation, it's definitely good enough to tell the difference between "a" and "an" before a word. Trying to account for every possible way your users will interact with Alexa, combined with local dialects, slang, and accents, can be quite a challenge, but you'll want to at least get the basics.

The next two files you'll want to familiarize yourself with are the **index.js** and **AlexaSkill.js** files, which are located in the **src** folders of each of the sample projects.

The index.js file from the HelloWorld sample is pretty straightforward and contains a few pieces that you'll need for the C# implementation later, so let's take a look at that one first.

At the beginning of the file, you'll find the APP_ID and the AlexaSkill library references, which you'll need to make use of once you have the skill defined with Amazon.

```
var APP_ID = undefined;
//replace with "amzn1.echo-sdk-ams.app.[your-unique-
value-here]";

var AlexaSkill = require('./AlexaSkill');
```

The onLaunch event greets users of your skill, and optionally provides them a brief synopsis of how they can expect to interact with it. Make note of the speechOutput and repromptText variables.

```
HelloWorld.prototype.eventHandlers.onLaunch
   = function (launchRequest, session, response)
{
   console.log("HelloWorld onLaunch requestId: "
      + launchRequest.requestId + ", sessionId: "
      + session.sessionId);

   var speechOutput = "Welcome to the Alexa Skills
      Kit, you can say hello";

    var repromptText = "You can say hello";

    response.ask(speechOutput, repromptText);
};
```

Next up are some session-related event handlers that should look familiar to anybody who's done some Web development in the past. Most Alexa sessions last for a single interaction, but it's possible, depending on the skill, for a session to last for four or five interactions.

The onSessionStarted event gives you an opportunity to do any necessary initialization prior to launching your skill.

```
HelloWorld.prototype.eventHandlers.onSessionStarted
   = function (sessionStartedRequest, session)
{
   console.log("HelloWorld onSessionStarted requestId:
      " + sessionStartedRequest.requestId
        + ", sessionId: " + session.sessionId);

    // any initialization logic goes here

};
```

The onSessionEnded event allows you an opportunity to add code for any necessary cleanup when the skill is finished executing.

```
HelloWorld.prototype.eventHandlers.onSessionEnded
   = function (sessionEndedRequest, session) {
```

Deploying Your Alexa Skill as an Azure API App.

In this article, I mention deploying my sample Alexa Skill as an Azure API app.

Unfortunately, going into the details of Azure or Web publishing is beyond the scope of this article.

Fortunately, Microsoft has an excellent article on publishing your solution as an Azure API app, which I encourage you to read: http://bit.ly/2hvH5NW.

```
    console.log("HelloWorld onSessionEnded requestId: "
      + sessionEndedRequest.requestId
      + ", sessionId: " + session.sessionId);

    // any cleanup logic goes here

};
```

Lastly, the intentHandlers method registers the intents that are defined in the IntentSchema.json file, and tell Alexa how to respond to them. Make note of the tellWithCard() method, which allows Alexa to respond with additional information, or even graphical content, via the Amazon Alexa app on your smart phone.

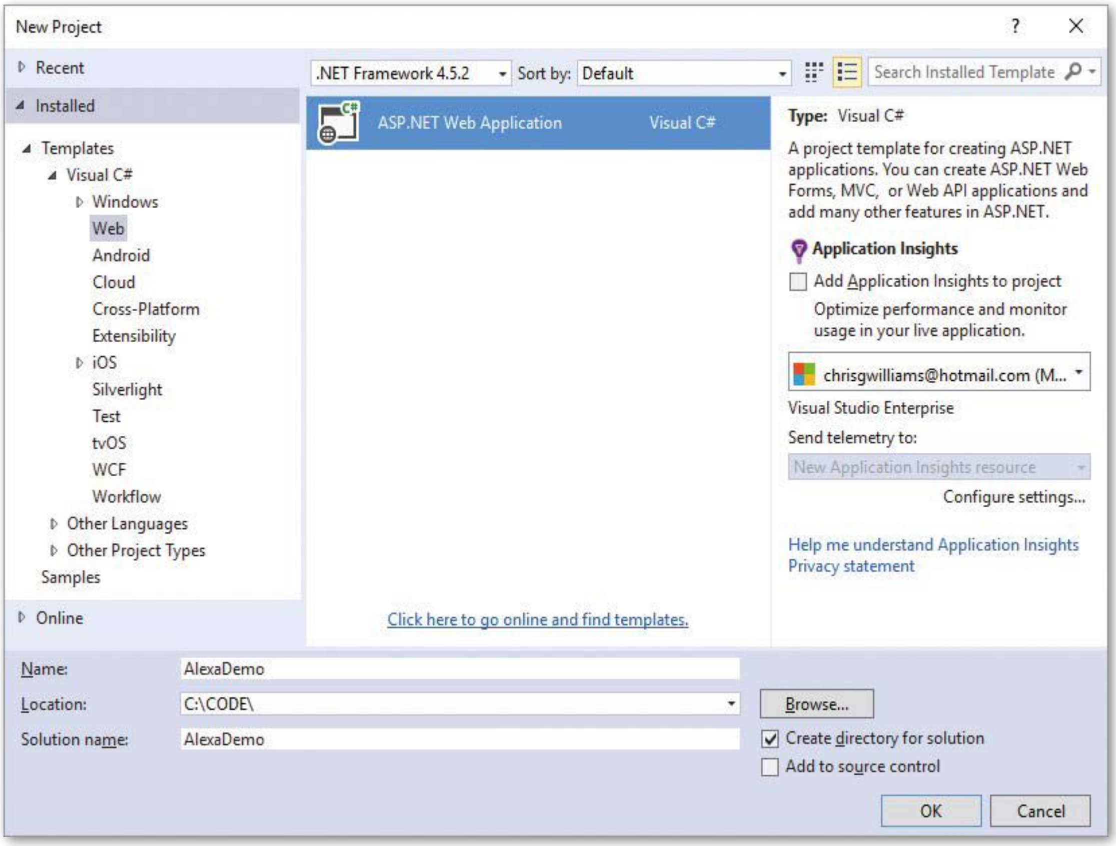
```
HelloWorld.prototype.intentHandlers = {
   // register custom intent handlers

   "HelloWorldIntent": function (intent, session,
      response) {
      response.tellWithCard("Hello World!", "Hello
         World", "Hello World!");
    },

    "AMAZON.HelpIntent": function (intent, session,
       response) {
       response.ask("You can say hello to me!", "You
          can say hello to me!");
    }
};
```

Most of the **index.js** file for the MinecraftHelper is very similar to this one, so I won't include it all here (see **Listing 1**), but the intenthandler section is a bit more involved, and is worth taking a look at, especially the section where it shows the simultaneous use of speech output and piping the same information to a card in the Alexa app.

The next file to take a look at is **AlexaSkill.js**, also located in the src directory of each sample project. This class serves as the base class for the index.js file, and contains the plumbing for the request, event, and intent handlers. There's nothing specific to your skill here, and each instance of this file is the same from one project to the next, so I won't spend any further time on that one right now.

There's also a **recipes.js** file, which is specific to the MinecraftHelper project. I won't include the whole thing here, because it isn't really relevant to what we're doing (and it's a huge file), but I will add a small snippet, just so you can see how external data can be referenced in the typical key/value or dictionary style:

```
module.exports = {
   "snow golem": "A snow golem can be created by
   placing a pumpkin on top of two snow blocks
   on the ground.",
   "glass": "Glass can be obtained by smelting
   sand in a furnace.",
   "boat": "A boat can be obtained by making a U
   shape with five wooden planks.",
   "hay bale": "A hay bale can be crafted by
   placing wheat in a 3 by 3 grid in a crafting
   table.",
   "clay": "Clay is obtained by breaking clay
   blocks with a non silk touch tool.",
```



Type: Visual C#

A project template for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

Application Insights

☐ Add Application Insights to project

Optimize performance and monitor usage in your live application.

chrisgwilliams@hotmail.com (M...

Visual Studio Enterprise

Send telemetry to:

New Application Insights resource

Name: AlexaDemo

Location: C:\CODE\

Solution name: AlexaDemo

☑ Create directory for solution

☐ Add to source control

Figure 3: Create a new Web Application project.

Listing 1: intentHandler for MinecraftHelper sample

```
HowTo.prototype.intentHandlers = {
   "RecipeIntent": function (intent, session, response) {
      var itemSlot = intent.slots.Item, itemName;

      if (itemSlot && itemSlot.value) {
         itemName = itemSlot.value.toLowerCase();
      }

      var cardTitle = "Recipe for " + itemName,
         recipe = recipes[itemName],
         speechOutput,
         repromptOutput;

      if (recipe) {
         speechOutput = {
            speech: recipe,
            type: AlexaSkill.speechOutputType.PLAIN_TEXT
         };

         response.tellWithCard(speechOutput, cardTitle, recipe);
      } else {
         var speech;

         if (itemName) {
            speech = "I'm sorry, I currently do not know the
               recipe for " + itemName + ". What else can I help
               with?";
         } else {
            speech = "I'm sorry, I currently do not know that
               recipe. What else can I help with?";
         }

         speechOutput = {
            speech: speech,
            type: AlexaSkill.speechOutputType.PLAIN_TEXT
         };

         repromptOutput = {
            speech: "What else can I help with?",
            type: AlexaSkill.speechOutputType.PLAIN_TEXT
         };

         response.ask(speechOutput, repromptOutput);
      }
   },

   "AMAZON.StopIntent": function (intent, session, response) {
      var speechOutput = "Goodbye";
      response.tell(speechOutput);
   },

   "AMAZON.CancelIntent": function (intent, session, response) {
      var speechOutput = "Goodbye";
      response.tell(speechOutput);
   },

   "AMAZON.HelpIntent": function (intent, session, response) {
      var speechText = "You can ask questions such as, what's the
         recipe, or, you can say exit... Now, what can I help you
         with?";

      var repromptText = "You can say things like, what's the
         recipe, or you can say exit... Now, what can I help you
         with?";

      var speechOutput = {
         speech: speechText,
         type: AlexaSkill.speechOutputType.PLAIN_TEXT
      };

      var repromptOutput = {
         speech: repromptText,
         type: AlexaSkill.speechOutputType.PLAIN_TEXT
      };

      response.ask(speechOutput, repromptOutput);
   }
};
```

```
   "bowl": "A bowl can be crafted by placing
   three wooden planks in a v shape in a crafting
   table."
};
```

The last file to look at in each of the sample projects is the README.md markdown file, which provides step-by-step instructions on how to deploy the project to AWS Lambda and setup the skill. This file is essential if you plan to do a test deployment of any of the sample projects.

At this point, you've looked at everything relevant that goes into the sample projects, so it's time to get started making your own Alexa Skill in C# and Visual Studio.

Time to Write Some Code

Start by creating a new ASP.NET Web Application in Visual Studio 2015 (see **Figure 3**). I'm using Visual Studio Community 2015 because it's free, but if you have the Professional or Enterprise editions, that's cool too. Name your project whatever you like (I'm naming mine AlexaDemo) and uncheck the box for Application Insights.

In the next dialog box, select the Web API template (see **Figure 4**) and unselect the "Host in the cloud" checkbox. I'm using the Web API template because it provides nearly everything I need right out of the box, with a minimum of additional work required. This makes it really easy to get up and running quickly.

Hit OK and wait a couple moments. If all goes well, you'll have the Project_Readme file onscreen, congratulating you for successfully creating a new project. Take a moment to bask in all that glory, then close the window, and push up your sleeves. It's time to do some work.

In your Solution Explorer under the AlexaDemo project, create a new folder and call it **SpeechAssets**. This is where you'll put the IntentSchema.js and SampleUtterances.txt files.

1. Add a new JS file to the folder and call it **IntentSchema.json**.
2. Add a new Text file (if you can't find the file type, look under General) to the folder and name it **SampleUtterances.txt**.

Once you've done that, it should look like **Figure 5**.

Next, open up File Explorer and dive into the helloWorld\speechAssets folder. Open up the IntentSchema.json and

SampleUtterances.txt files and copy/paste their contents into the empty files of the same name you created a moment ago. Make sure to save them.

You'll come back to these shortly. The next thing you need is a controller class, because you're using Web API. Right-click on the Controllers folder and add the "Web API 2 Controller - Empty" scaffold, as seen in **Figure 6**.

Give it a name (AlexaDemoController.cs sounds good) and hit the Add button. After a few seconds, you'll be staring at an empty controller class.

Before you start writing the controller, take a look at **Listing 2**, which is a slightly modified version of the **Sample Response** found in the ASK Interface Reference docs (it's here: https://developer.amazon.com/public/solutions/alexa/alexa-skills-kit/docs/alexa-skills-kit-interface-reference). This represents the Response object that your service returns to the Alexa Voice Service.

I'm not using any session attributes in this sample, but the sessionAttribute collection is where you map any key-value pairs that need to be persisted within an Alexa session.

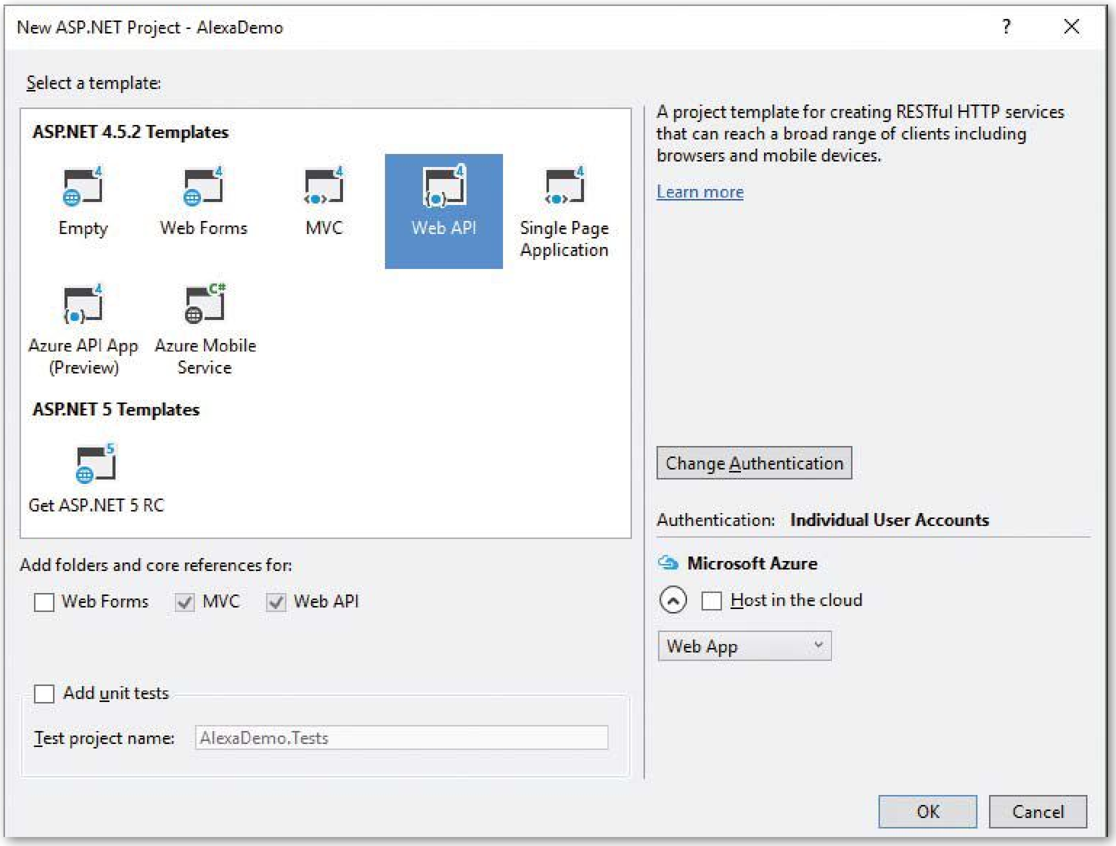
You'll notice that within the "response" section, there are three entries: outputSpeech, card and reprompt. The text in the outputSpeech block is what the AVS converts back into speech for Alexa to read to you, and the card block content is what gets sent to the Alexa Companion App on your phone or tablet. The reprompt block contains what Alexa says to you if you don't respond to the initial response after activating your skill.

Be aware of the following size limitations when crafting your response:

Learn More About Integrating Alexa Voice Service Everywhere.

If you really want to go beyond developing for Amazon Echo devices, start integrating Alexa into other devices or building your own hardware!

Read "Getting Started with the Alexa Voice Service" for an excellent walkthrough of the process: http://amzn.to/1WwsJYs.



A project template for creating RESTful HTTP services that can reach a broad range of clients including browsers and mobile devices.

Authentication: **Individual User Accounts**

Microsoft Azure

Host in the cloud [ ]

Add folders and core references for: Web Forms [ ] MVC [✓] Web API [✓]

Add unit tests [ ]

Test project name: AlexaDemo.Tests

Figure 4: The Web API Template

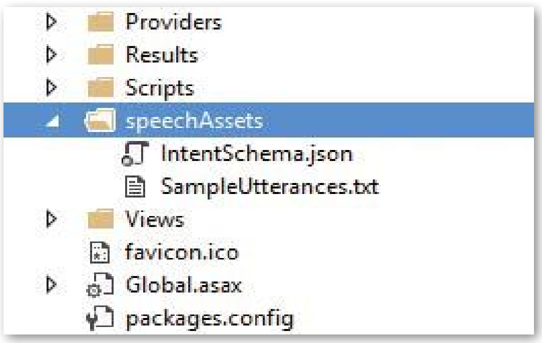
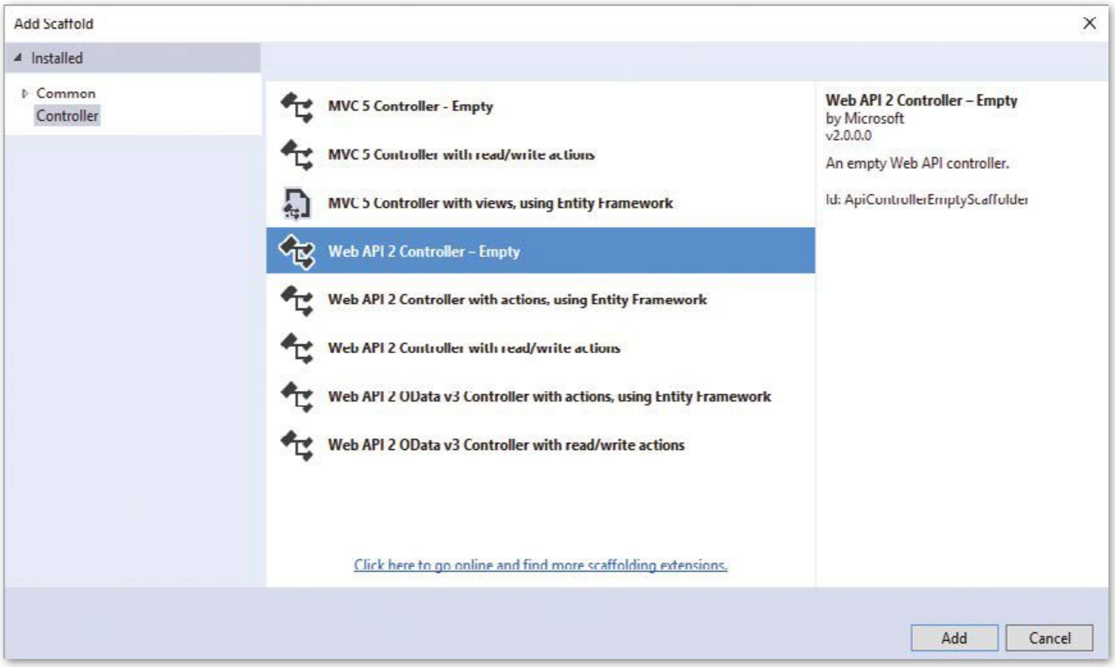


Figure 5: The speechAssets folder

- The outputSpeech response can't exceed 8000 characters.
- All of the text included in a card can't exceed 8000 characters. This includes the title, content, text, and image URLs.
- An image URL (smallImageUrl or largeImageUrl) can't exceed 2000 characters.
- The token included in an audioItem.stream for the AudioPlayer.Play directive can't exceed 1024 characters.
- The URL included in an audioItem.stream for the AudioPlayer.Play directive can't exceed 8000 characters.
- The total size of your response can't exceed 24 kilobytes.



Add Scaffold
Installed
Common
Controller
MVC 5 Controller - Empty
MVC 5 Controller with read/write actions
MVC 5 Controller with views, using Entity Framework
Web API 2 Controller – Empty
Web API 2 Controller with actions, using Entity Framework
Web API 2 Controller with read/write actions
Web API 2 OData v3 Controller with actions, using Entity Framework
Web API 2 OData v3 Controller with read/write actions
Click here to go online and find more scaffolding extensions.
Add
Cancel

Web API 2 Controller – Empty
by Microsoft
v2.0.0.0
An empty Web API controller.
Id: ApiControllerEmptyScaffolder

Figure 6: Add an empty controller scaffold to your project.

If your response exceeds these limits, the Alexa service returns an error at deployment time.

Now that you have an idea of what a proper response should look like, it's time to jump back in to Visual Studio and add some code to the controller. Add this dynamic method signature inside the AlexaDemoController class:

```
[HttpPost, Route("api/alexa/demo")]
public dynamic AlexaDemo(dynamic request) {

}
```

If you're writing a single Web service to support additional speech enabled devices beyond Alexa, you can include any additional SSML tags that the other devices support in the response and Alexa ignores anything she doesn't recognize.

The Alexa Voice Service requires the message to be sent as a POST (not a GET) request and the Route attribute is how the request will get to the right API call in the service. Of course, you could create a C# class to strongly type the return object, but this dynamic approach works just as well.

In the message, in addition to the response, you'll need to include a version number and session attributes (even if empty).

Listing 2: Modified Sample Response from ASK Reference Docs

```
{
  "version": "1.0",
  "sessionAttributes": {},
  "response":
  {
    "outputSpeech":
    {
      "type": "PlainText",
      "text": "Hello World, from the Alexa Demo"
    },
    "card":
    {
      "type": "Simple",
      "title": "Alexa Demo",
      "content": "This is the Alexa demo."
    },
    "reprompt":
    {
      "outputSpeech":
      {
        "type": "PlainText",
        "text": "Can I help you with anything else?"
      }
    },
    "shouldEndSession": false
  }
}
```

I'm not using any session attribute in this example, but if you were to add some, they would be in key/value pairs, and would be defined like this:

```
sessionAttributes = new {
    "string": object,
    "string": object
}
```

The same rules that apply to sessions in Web pages are relevant here as well; don't assume they'll always be there because the session could expire at any time, and don't put anything sensitive in them.

Inside the AlexaDemo() method, add the following code:

```
return new {
    version = "1.0",
    sessionAttributes = new { },
    response = new {
        outputSpeech = new {
            type = "PlainText",
            text = "Welcome to the Alexa Demo"
        },
        card = new {
            type = "Simple",
            title = "Alexa Demo",
            content = "Welcome to the Alexa Demo"
        },
        shouldEndSession = true
    }
};
```

The shouldEndSession property of the response object is something I haven't talked about yet. In a skill that requires multiple interactions with Alexa, you set this to **false**, to preserve any session attributes until the end of the interaction, but in this example there's no need to do this, so I've set it to **true**.

Within the outputSpeech block, you have a type of **PlainText**, which is exactly what it sounds like. Whatever you put here is exactly what Alexa will read back to you.

The Alexa Voice Service also supports another type, called Speech Synthesis Markup Language (SSML). SSML is used when you need to control how Alexa generates the speech, such as specific inflection via phonemes, adding a longer pause between phrases, or even injecting an MP3 clip in the response.

Amazon didn't create the SSML standard, but they do support a subset of it. You can find the entire list of Alexa-supported SSML tags here: http://amzn.to/1OaLmAZ.

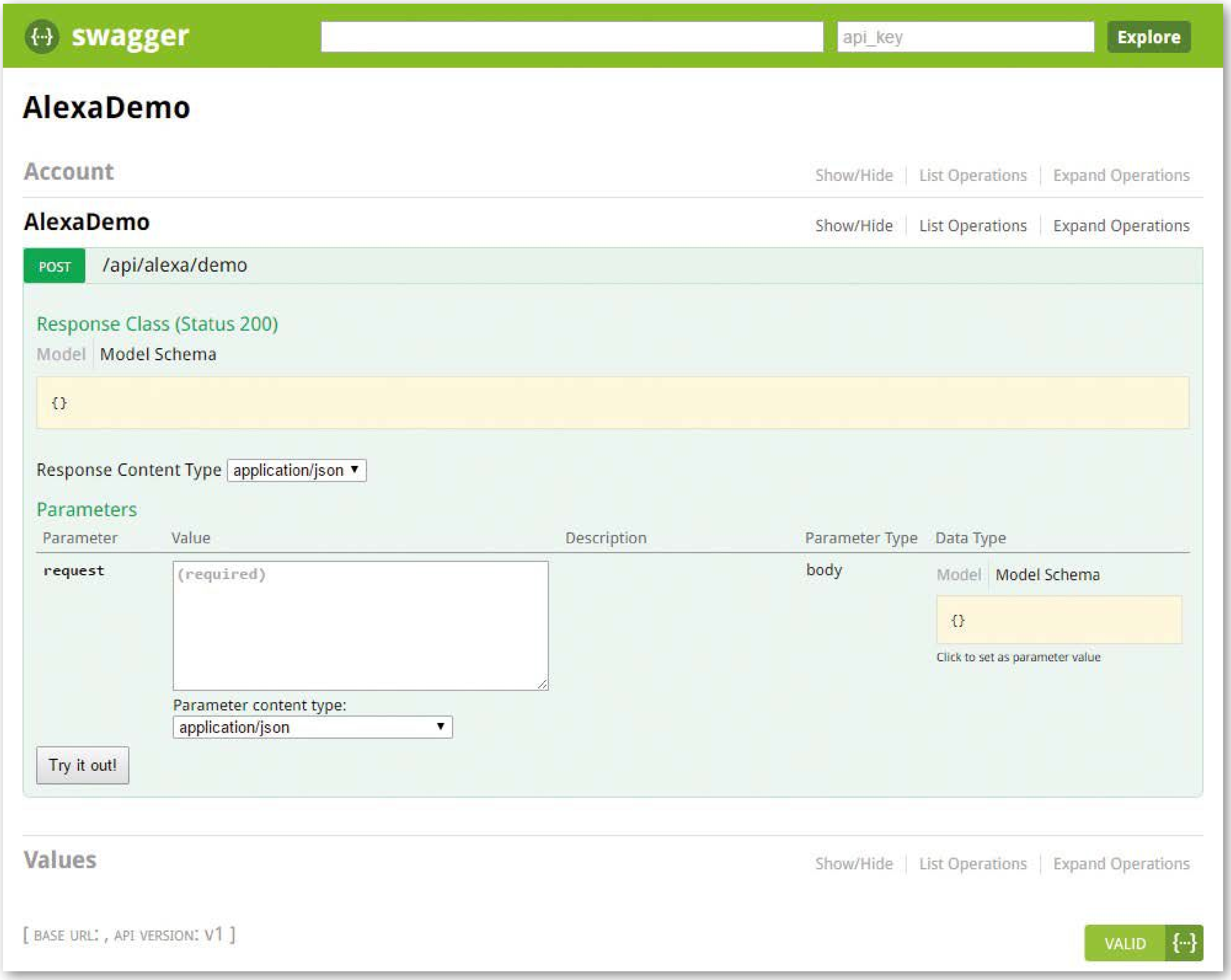
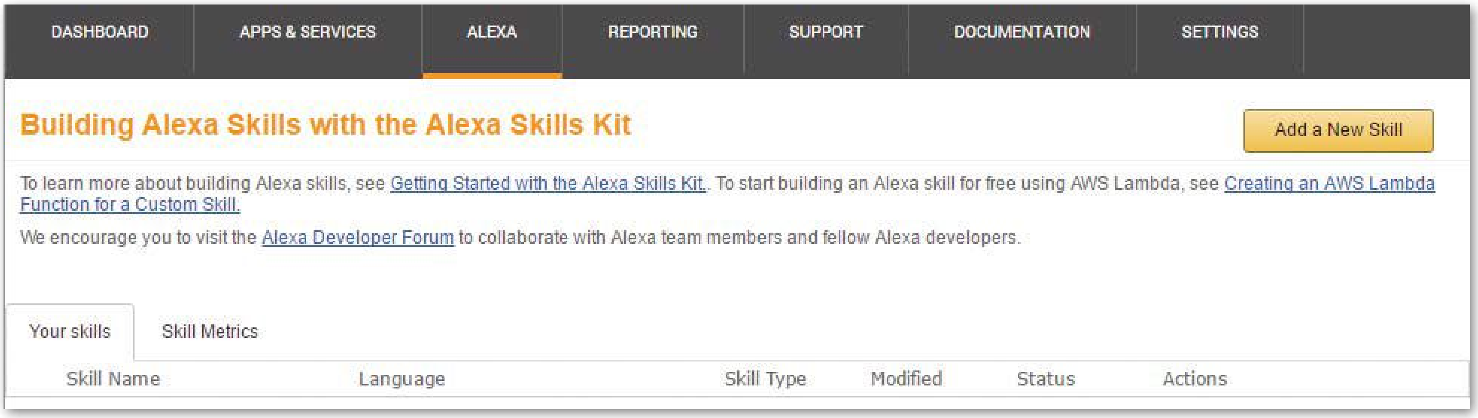


Figure 7: AlexaDemo with Swagger!



Building Alexa Skills with the Alexa Skills Kit

To learn more about building Alexa skills, see Getting Started with the Alexa Skills Kit. To start building an Alexa skill for free using AWS Lambda, see Creating an AWS Lambda Function for a Custom Skill.

We encourage you to visit the Alexa Developer Forum to collaborate with Alexa team members and fellow Alexa developers.

Skill Name	Language	Skill Type	Modified	Status	Actions

Figure 8: The Alexa Skills Page

If you're writing a single Web service to support additional speech enabled devices beyond Alexa, you can include any additional SSML tags that the other devices support in the response and Alexa ignores anything she doesn't recognize.

The following snippet shows what your outputSpeech block looks like with SSML instead of plain text and with an embedded sound clip:

```
"outputSpeech" = new {
    type= "SSML",
    ssml= "<speak>This output speech uses SSML.
             <audio src=http://www.mp3.com/file.mp3 />
           </speak>"
}
```

If you're using SSML instead of plain text, there are some limits you should be aware of when crafting your service's response to Alexa:

- The entire outputSpeech response may not be greater than 8000 characters.
- Card text may not exceed 8000 characters, including image URLs.
- Image URLs may not exceed 2000 characters.
- The total size of your response message may not exceed 24K.
- These are the limitations most relevant to what I'm doing here, but there are more than this, so be sure to look at all of them (here: http://amzn.to/1UcAgxa) if you plan to work with SSML.

Congratulations! You're done with coding the controller. In the next section, I cover deploying and testing your Alexa skill.

Deploying and Testing Your Service

In order to deploy and test your Alexa Skill, you'll have to publish it somewhere. I've published mine as an Azure API app (which you can find at http://bit.ly/2gXh4In), but you're free to put yours anywhere that's publicly accessible on the Web or the cloud. There are benefits (and drawbacks) no matter which you choose, so I'm leaving that up to you.

If you visit the previous link, you'll see something that looks like **Figure 7**, and if you aren't familiar with Swagger, you're in for a real treat.

Swagger is a great set of free open source tools that help you build and test your RESTful APIs. You may have heard of Postman, which is another great tool. In this article, I'm using Swagger (mostly because you get it automatically when publishing to Azure API apps, which is even better).

To test the service, type anything (i.e., "hi" or "hello" or whatever) in the request box and click the "Try it out!" button. You'll see the Response Body, which matches the response object you defined earlier, along with a Response Code of 200, and the Response Headers, which you can see below:

```
{
  "pragma": "no-cache",
  "date": "Sun, 18 Dec 2016 19:27:47 GMT",
  "content-encoding": "gzip",
  "server": "Microsoft-IIS/8.0",
  "x-aspnet-version": "4.0.30319",
  "x-powered-by": "ASP.NET",
  "vary": "Accept-Encoding",
  "content-type": "application/json; charset=utf-8",
  "cache-control": "no-cache",
  "x-ms-proxy-outgoing-newurl": "https://microsoft-
   apiapp82820479ea5048298e6d4a3969eb78f7
   .azurewebsites.net/api/alexa/demo",
  "content-length": "269",
  "expires": "-1"
}
```

Now that you've tested the service, it's time to visit the Amazon Developer Portal and add your new skill.

Setting Up an Alexa Skill

In your browser of choice, visit http://developer.amazon.com/ask and sign in (or sign up) to get to the Developer Console.

If you're signing up for the first time, be aware that whatever account you use going forward should be the one that's paired up to your Amazon Echo device and Marketplace Account (if you have one).

Once you're in, click on ALEXA in the top menu. You should see (at least) two options available. Click the Get Started button under the Alexa Skills Kit and you'll be taken to the Skills page (shown in **Figure 8**), which is where you will add, edit, and track the various skills you've submitted.

Click the Add a New Skill button, and you are presented with some fields to fill out in order to create your Alexa Skill.

To create the skill for the Alexa Demo, start by filling in the Skill Information screen (as shown in **Figure 9.**) Do the following:

1. Select the **Custom Interaction Model** radio button.
2. Select the Language of your skill (default is English U.S.)
3. Type in the Name of the skill (i.e., Alexa Demo)
4. Type in the Invocation Name (what you say to activate the skill)
5. Click the Next button.

On the next screen (**Figure 10**), define the interaction model.

1. Switch back over to Visual Studio, copy the JSON contents of your intentSchema.js file, and then paste it into the text box labeled **Intent Schema**.
2. You haven't defined any Custom Slot Types in this skill, so you can skip that section.
3. Do the same thing for the SampleUtterances.txt file, pasting it into the Sample Utterances text box.
4. Click the **Next** button.

In the Configuration section (see **Figure 11**), provide information about your service endpoint.

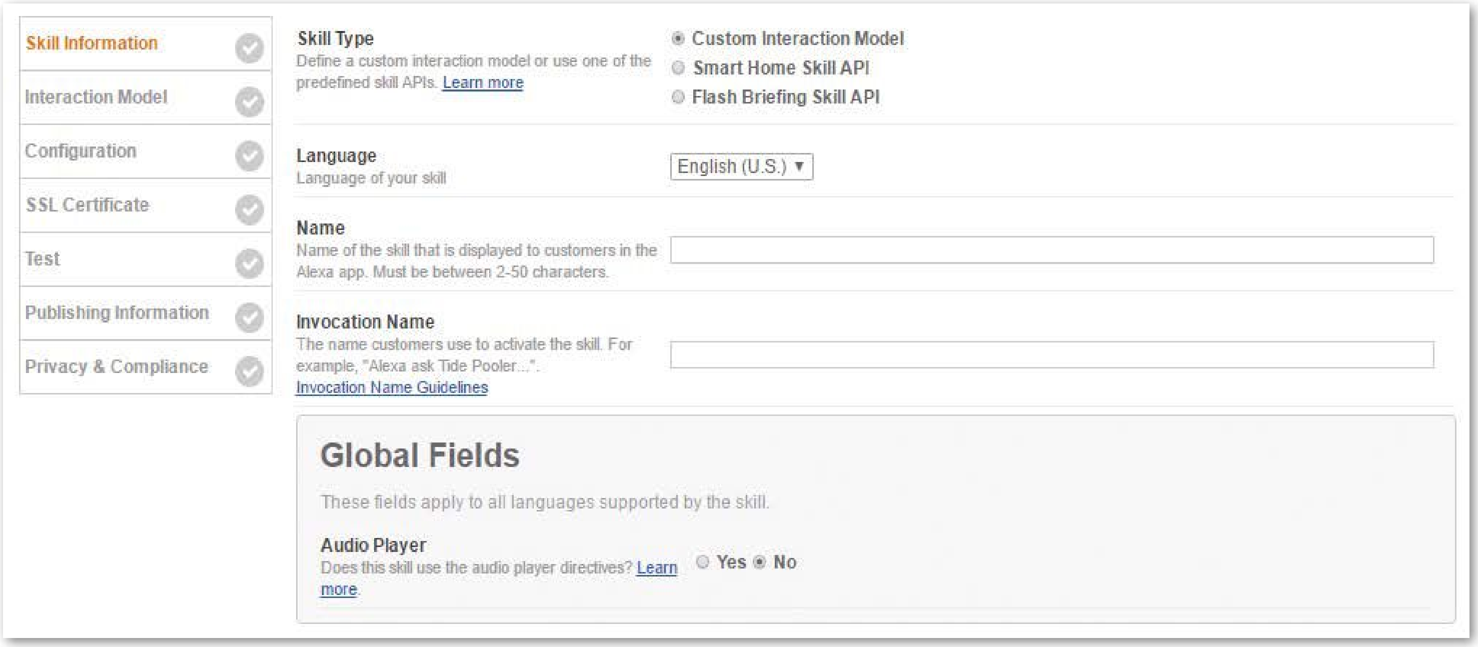
1. Select the HTTPS radio button.
2. Select the geographical region closest to you (North America or Europe)
3. Enter the URL of your service endpoint (this is wherever you hosted it, including the "api/alexa/demo".)
4. Click the Next button.

In the SLL Certificate section (**Figure 12**), provide the required certificate information.

1. If you're hosting your service on your own website and it has a certificate, select "My development endpoint has a certificate from a trusted certificate authority."
2. If you're hosting your service in Azure (like me), you have a built in certificate already, so select "My development endpoint is a sub-domain of a domain that has a wildcard certificate from a certificate authority."
3. If neither of those apply, you have the option of selecting "I will upload a self-signed certificate in X.509 format." If you don't know how to create your own cert, Amazon provides you with a link next to this option to walk you through the process.
4. Click the **Next** button.

Next up is the Test section (**Figure 13**). There are no step-by-step instructions here, just some handy tools to play with:

- You can set your skill to Enabled or Disabled. It's set to **Enabled** by default.
- In the Voice Simulator text box, you can type anything you want, just to get an idea of what your skill sounds like coming from Alexa. This is super handy if you don't actually HAVE an Amazon Echo to test with. Also, this is a great place to practice some of those SSML tags I mentioned earlier. (SSML Reference: http://amzn.to/2i0Sh2a)
- In the Service Simulator box, you can enter an utterance to test your endpoint. Let's do that now, by typing **Hi** in the "Enter Utterance" box and clicking the **Ask Alexa Demo** button.
- Once you click the button, the Service Simulator displays the Service Request and Service Response in JSON. If you don't get anything back, check the URL of your HTTPS endpoint, as Alexa may be having a problem reaching your service.
- You've seen the response object a few times by now, so I won't include it here again, but you should def-



Skill Type
Define a custom interaction model or use one of the predefined skill APIs. Learn more

[✓ Custom Interaction Model] Smart Home Skill API Flash Briefing Skill API

Language
Language of your skill

English (U.S.)

Name
Name of the skill that is displayed to customers in the Alexa app. Must be between 2-50 characters.

Invocation Name
The name customers use to activate the skill. For example, "Alexa ask Tide Pooler...".
Invocation Name Guidelines

Global Fields

These fields apply to all languages supported by the skill.

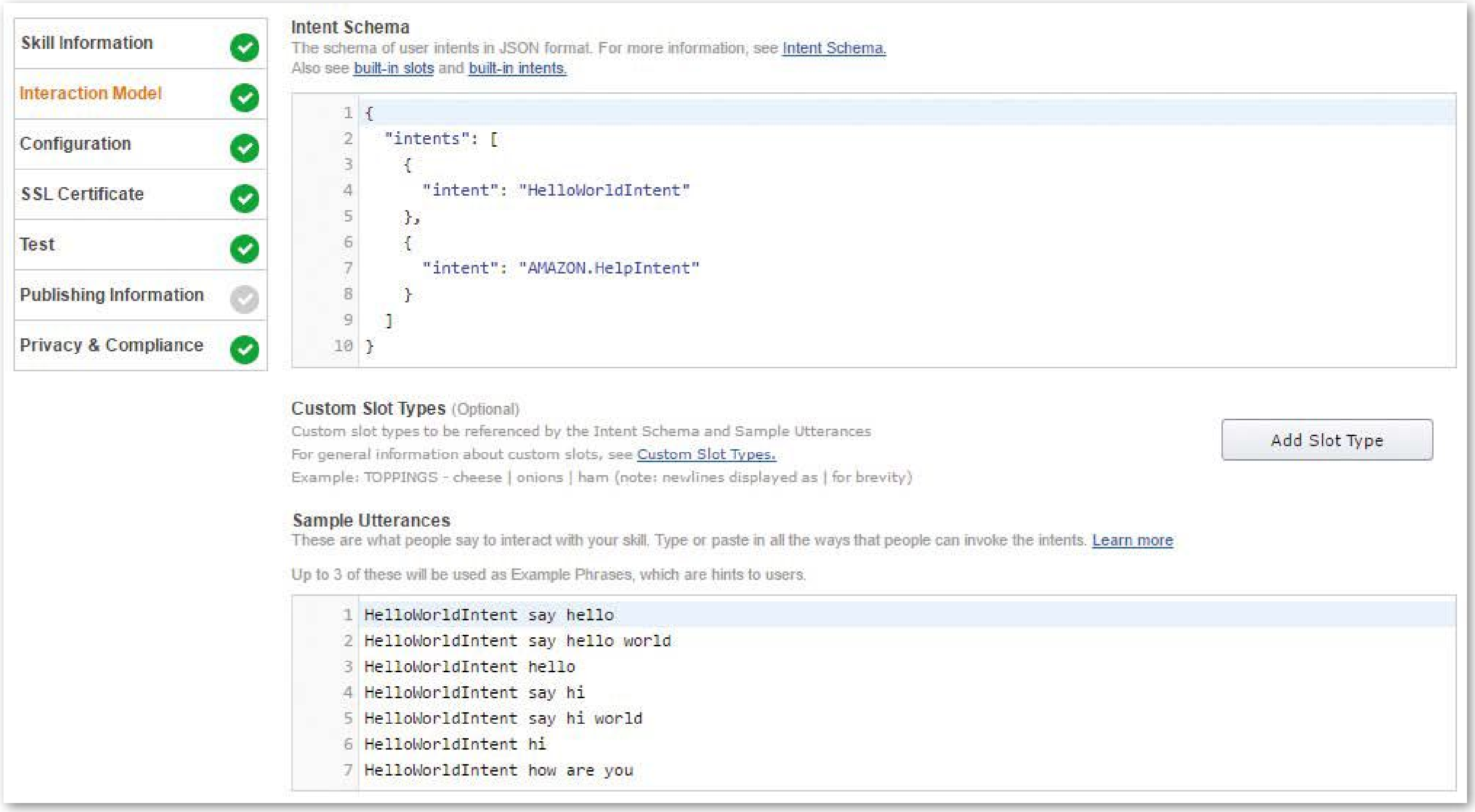
Audio Player
Does this skill use the audio player directives? Learn more.

Yes [✓ No]

Figure 9: The Skill Information Tab

initely take a minute to look at the **Service Request** (either in the simulator or in **Listing 3**.) There's a lot of good information in there, especially if you need to debug your endpoint. Among other things, it shows what Intent is being called, along with any parameters (slots) being passed along with it.

- Also, under the Service Response section, you can click the Listen button to hear Alexa read the relevant portion of the service response out loud.
- When you're finished in this section, go ahead and click the Next button (again).



Intent Schema
The schema of user intents in JSON format. For more information, see Intent Schema.
Also see built-in slots and built-in intents.

```
{
  "intents": [
    {
      "intent": "HelloWorldIntent"
    },
    {
      "intent": "AMAZON.HelpIntent"
    }
  ]
}
```

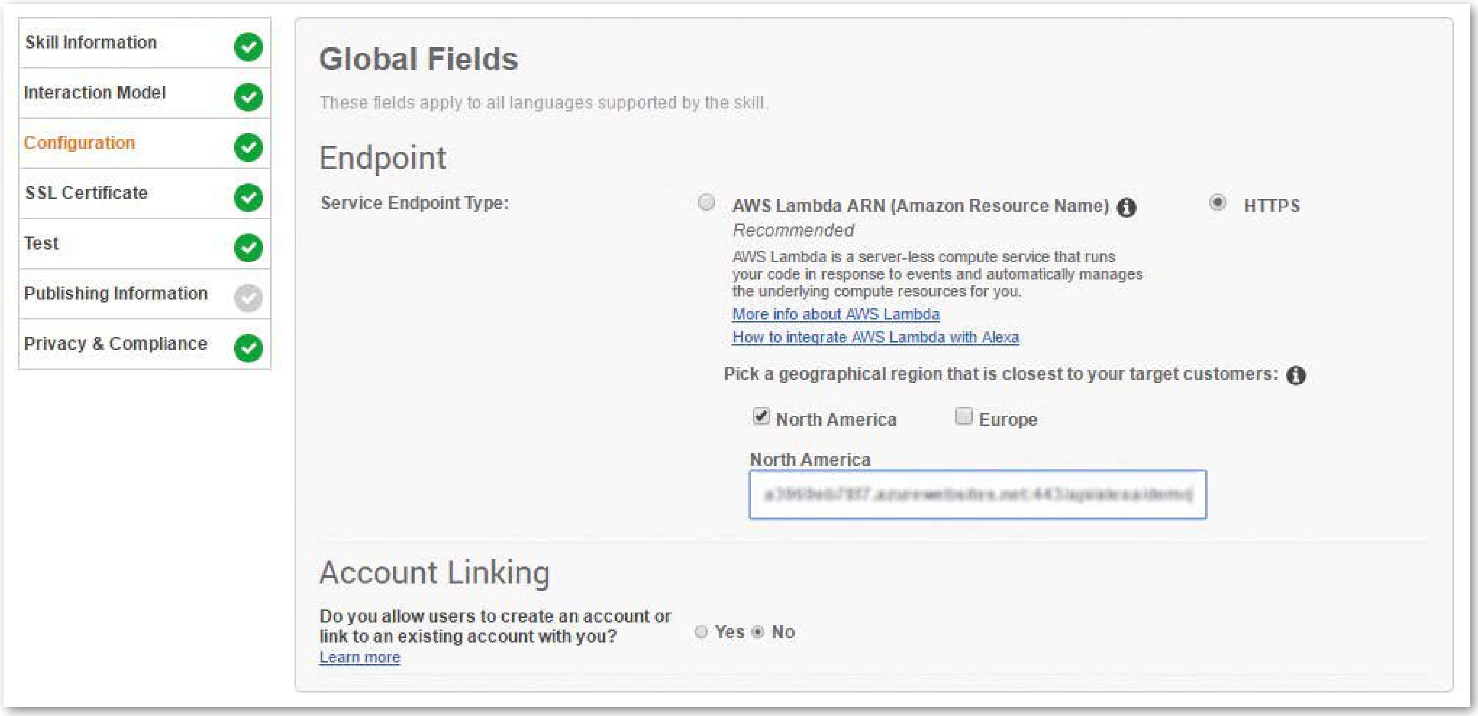
Custom Slot Types (Optional)
Custom slot types to be referenced by the Intent Schema and Sample Utterances
For general information about custom slots, see Custom Slot Types.
Example: TOPPINGS - cheese | onions | ham (note: newlines displayed as | for brevity)

Sample Utterances
These are what people say to interact with your skill. Type or paste in all the ways that people can invoke the intents. Learn more

Up to 3 of these will be used as Example Phrases, which are hints to users.

```
HelloWorldIntent say hello
HelloWorldIntent say hello world
HelloWorldIntent hello
HelloWorldIntent say hi
HelloWorldIntent say hi world
HelloWorldIntent hi
HelloWorldIntent how are you
```

Figure 10: The Interaction Model Tab



Global Fields

These fields apply to all languages supported by the skill.

Endpoint

Service Endpoint Type: AWS Lambda ARN (Amazon Resource Name) *Recommended* [✓ HTTPS]

AWS Lambda is a server-less compute service that runs your code in response to events and automatically manages the underlying compute resources for you.
More info about AWS Lambda
How to integrate AWS Lambda with Alexa

Pick a geographical region that is closest to your target customers:

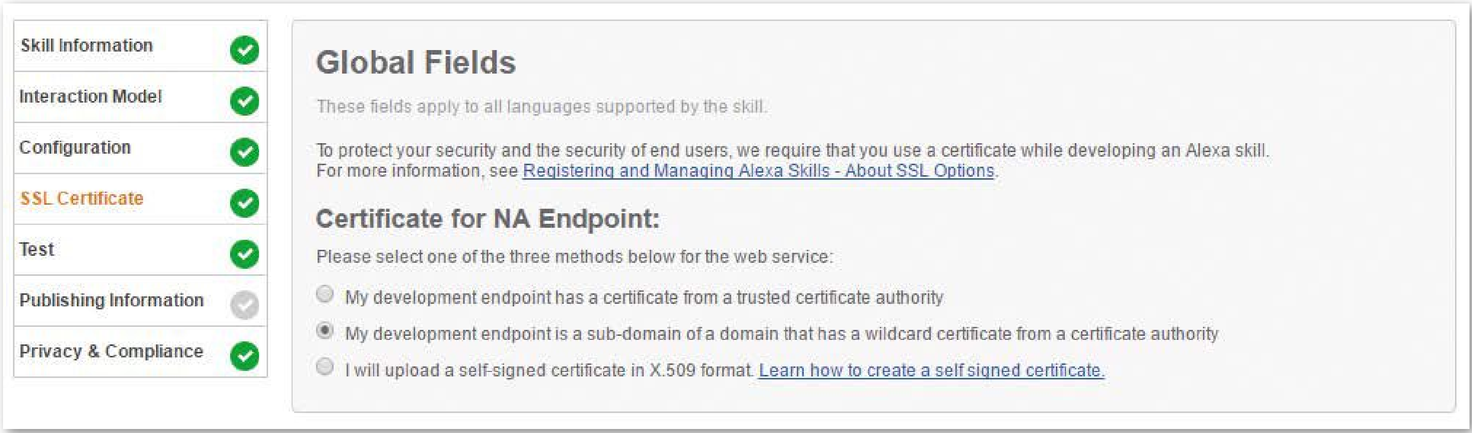
[✓ North America] Europe

North America
[illegible]

Account Linking

Do you allow users to create an account or link to an existing account with you? Learn more
Yes [✓ No]

Figure 11: The Configuration Tab


Skill Information ✓
Interaction Model ✓
Configuration ✓
SSL Certificate ✓
Test ✓
Publishing Information
Privacy & Compliance ✓

Global Fields

These fields apply to all languages supported by the skill.

To protect your security and the security of end users, we require that you use a certificate while developing an Alexa skill. For more information, see Registering and Managing Alexa Skills - About SSL Options.

Certificate for NA Endpoint:

Please select one of the three methods below for the web service:

- ○ My development endpoint has a certificate from a trusted certificate authority
- ● My development endpoint is a sub-domain of a domain that has a wildcard certificate from a certificate authority
- ○ I will upload a self-signed certificate in X.509 format. Learn how to create a self signed certificate.

Figure 12: The SSL Certificate Tab

Listing 3: Service Simulator Service Request

```
{
  "session": {
    "sessionId": "SessionId.6b864bd4-2f30-4873-a2c3-a34b23827903",
    "application": {
      "applicationId": "amzn1.ask.skill.f980f2fd-6573-4c33-a4cc-
      c8a8aa5e6fc4"
    },
    "attributes": {},
    "user": {
      "userId": "amzn1.ask.account.xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
    },
    "new": true
  },
  "request": {
    "type": "IntentRequest",
    "requestId": "EdwRequestId.b8c8bfc8-f03a-4b1f-91d1-
               0649514dd796",
    "locale": "en-US",
    "timestamp": "2016-12-18T21:48:59Z",
    "intent": {
      "name": "HelloWorldIntent",
      "slots": {}
    }
  },
  "version": "1.0"
}
```

At this point, your skill isn't published yet, but it *is* available for you to test locally on your Amazon Echo device. Just open the Alexa Companion App on your phone, tap the menu icon in the top left, tap Skills, and then tap the "Your Skills" link on the top right.

You're presented with an alphabetically sorted list of all the skills you've enabled on your Amazon Echo device, and because you named the skill "Alexa Demo," it should be at the top (or really close.) Go ahead and tap the skill and take a look at the detail page for the skill. It should already be enabled, but if not, just tap the big green Enable Skill button.

You'll notice that there's no icon and no description because you haven't added those yet. Also, nobody but you can see this skill, because it isn't published, but it'll work on your device, so go ahead and give it a try!

If you have an Amazon Echo device, make sure that you're in the correct profile and say this: "Alexa Tell AlexaDemo Hello" and Alexa responds with "Welcome to the Alexa demo."

Let's go back to the Developer Console and pick up where you left off, in the Publishing Information section.

I honestly don't recommend submitting this skill for certification. It will probably get rejected by Amazon because it doesn't really do much, and it doesn't fit any of the readymade categories (it's not a game or lifestyle skill, for example), but you can at least fill in the Short and Full Skill Description fields, along with the example phrases. If you've got an icon or logo handy, add those as well.

If you have an Amazon Echo device, say this: "Alexa Tell AlexaDemo Hello" and Alexa responds with "Welcome to the Alexa demo."

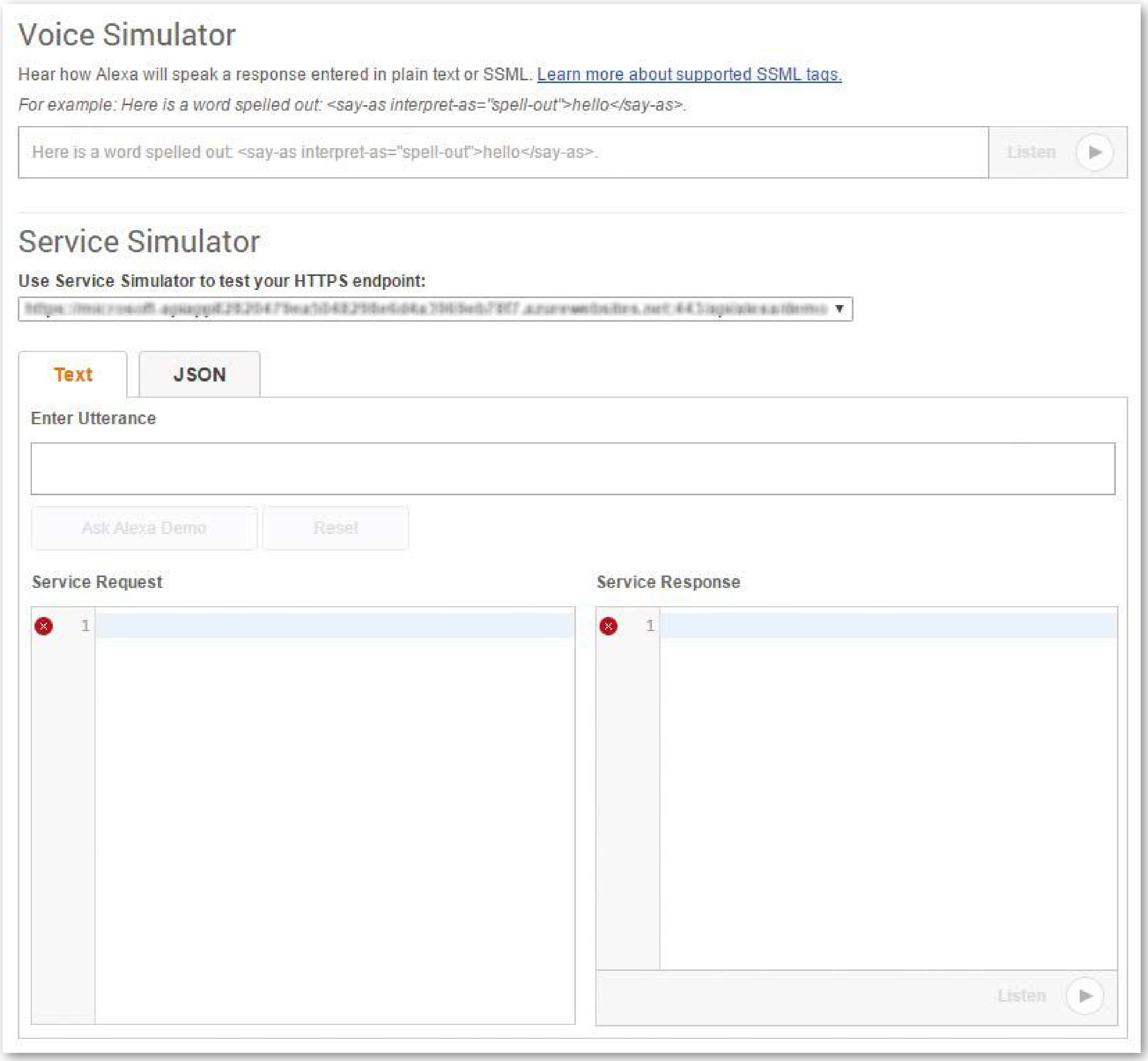
Once you hit the Save button, go look in your companion app again and you'll see that everything has been updated accordingly. Go ahead and click the Next button one more time.

The Privacy & Compliance section (**Figure 14**) asks you a few basic questions about your app, in terms of who it's targeted for, if it allows users to spend money, and if it collects any information about your users.

And with that, you're done. Go forth and make many awesome Alexa Skills.

For Further Thought

As I mentioned earlier, one of the really great things about the Amazon Voice Service is that you aren't limited to developing for the Amazon Echo. It's not a big leap from this device to connected home and car applications, or even pocket-sized devices, using a Raspberry Pi Zero and a Cellular piHat.



Hear how Alexa will speak a response entered in plain text or SSML. Learn more about supported SSML tags.

For example: Here is a word spelled out: <say-as interpret-as="spell-out">hello</say-as>.

Here is a word spelled out: <say-as interpret-as="spell-out">hello</say-as>.

Use Service Simulator to test your HTTPS endpoint:

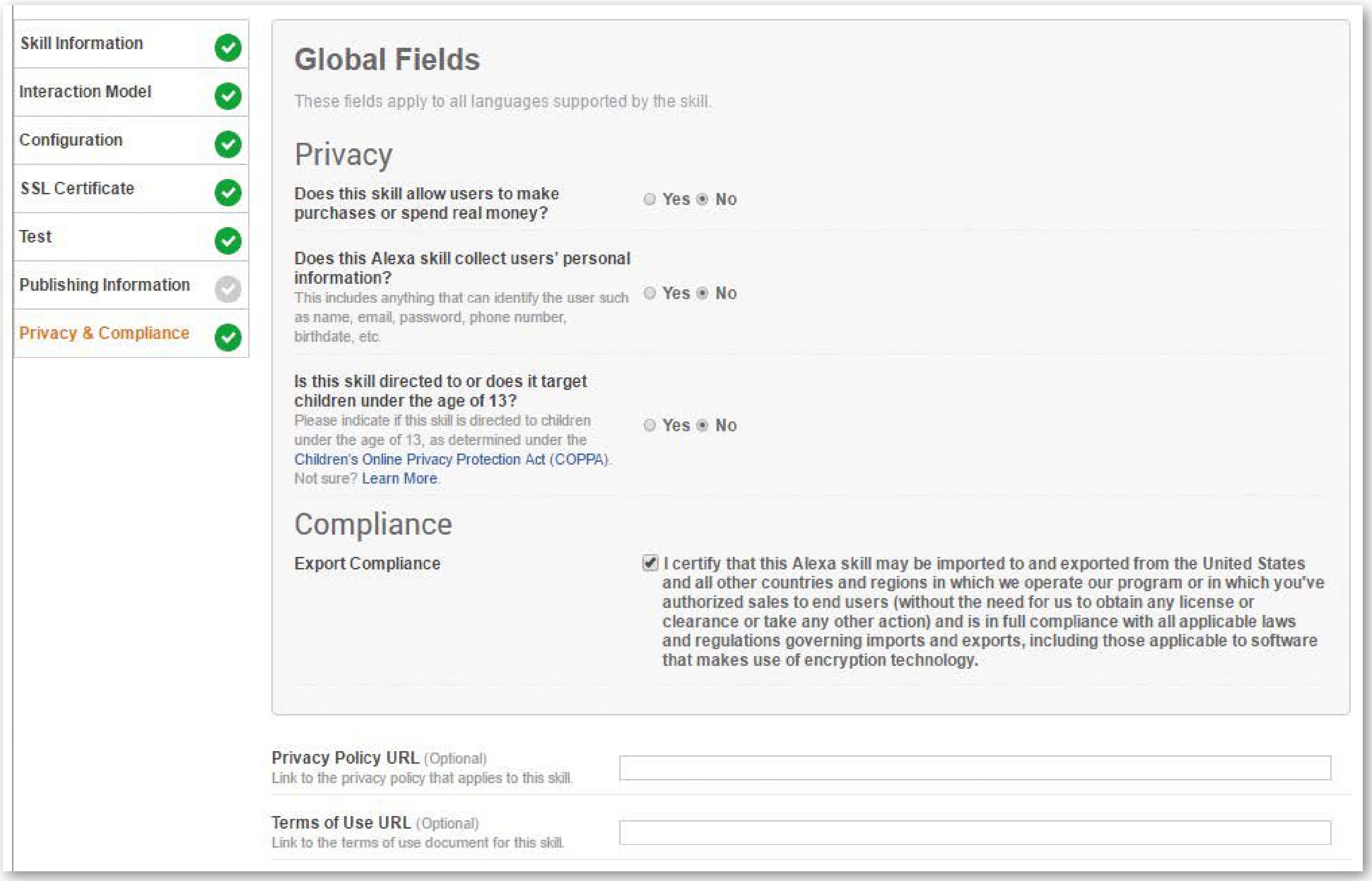
Figure 13: The Test Tab

Imagine having Alexa in your home, connected to a smart device hub that controls your lights, appliances, and home security, combined with a connected device in your car that responds to voice commands. Being able to say "Alexa Turn Off The Coffee Pot" while inside your house is pretty neat. Realizing that you left it on when you're 10 miles down the road and being able to say it in your car—and have Alexa turn off the pot—is a whole new level of awesome.

Additional Skills

In addition to the Custom Alexa Skill you just created, there are other APIs supported by Alexa that are worth investigating:

- **The Smart Home Skill API:** This API allows you to add skills to Alexa that include controlling any cloud connected device, such as the lighting, thermostat, appliances, smart plugs, and security features of your home. You can read more about this API at http://amzn.to/2hjAx1Z.
- **The Flash Briefing Skill API:** This API lets you add your own audio content feed to the Alexa Flash Briefing options via RSS. (If you have an Alexa-supported device and you haven't listened to a Flash Briefing yet, just say "Alexa give me my Flash Briefing" to get an idea of why this is useful.) Additional information about this API is available at http://amzn.to/2gUCbpY.



Skill Information	✓
Interaction Model	✓
Configuration	✓
SSL Certificate	✓
Test	✓
Publishing Information	
Privacy & Compliance	✓

Global Fields

These fields apply to all languages supported by the skill.

Privacy

Does this skill allow users to make purchases or spend real money? Yes [✓ No]

Does this Alexa skill collect users' personal information?
This includes anything that can identify the user such as name, email, password, phone number, birthdate, etc. Yes [✓ No]

Is this skill directed to or does it target children under the age of 13?
Please indicate if this skill is directed to children under the age of 13, as determined under the Children's Online Privacy Protection Act (COPPA). Not sure? Learn More Yes [✓ No]

Compliance

Export Compliance [✓] I certify that this Alexa skill may be imported to and exported from the United States and all other countries and regions in which we operate our program or in which you've authorized sales to end users (without the need for us to obtain any license or clearance or take any other action) and is in full compliance with all applicable laws and regulations governing imports and exports, including those applicable to software that makes use of encryption technology.

Privacy Policy URL (Optional)
Link to the privacy policy that applies to this skill.

Terms of Use URL (Optional)
Link to the terms of use document for this skill.

Figure 14: The Privacy & Compliance Tab

- **The List Skill API:** This is a new addition to the ASK and allows you to teach Alexa how to interface directly with your list applications, such as the Shopping List or To-Do List, so you don't have to build out a separate voice interaction model. You can learn all about this API at http://amzn.to/2hjD5gq.

Realizing that you left the coffee pot on when you're 10 miles down the road and telling Alexa to turn it off for you is a whole new level of awesome.

The Alexa Fund

The Alexa Fund provides up to $100 million in venture capital funding to fuel voice technology innovation. Amazon and a lot of other investors are betting big that human voice interaction will drive technology adoption in ways we have yet to even imagine.

In addition to funding, they also offer services such as product incubation, providing advice and resources to deliver your product to market.

If you're creating things that push the boundaries of voice technologies, whether it's software or hardware, you should take a look at The Alexa Fund (at https://developer.amazon.com/alexa-fund).

Chris G. Williams

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

Creating iMessages Apps

Apple introduced iMessages in June 2011 as the default messaging platform for iOS users; it's served well for Apple since then and now has millions of users. Unfortunately, iOS developers were never allowed to plug into the iMessages system and customize it according to their needs. This all changed in 2016 when Apple introduced the Messages framework in iOS 10, allowing developers to customize the look and feel of iMessages in the form of stickers and custom iMessages extensions using Swift 3.0.

Today, there are thousands of stickers and iMessages apps in the App Store and some even made it to the top ten paid apps. In this article, you're going to build couple of iMessages apps and look at the different methods and purposes for each approach.

Mohammad Azam
azamsharp@gmail.com
www.azamsharp.com
@azamsharp

Mohammad Azam works as an iOS Instructor at The Iron Yard. Azam trains and educates professionals on how to develop next generation iOS applications using Objective-C and Swift. Before joining The Iron Yard Azam worked as a Senior Mobile Developer at Blinds.com. He's been developing iOS applications since 2010 and has worked on more applications than he can remember.

Azam's current app, Vegetable Tree—Gardening Guide, is considered the best vegetable gardening app on the app store and was featured by Apple in 2013. He's a frequent speaker at Houston Tech Fest and Houston iPhone Meetup.

When not programming, Azam likes to spend his time with his beautiful family and planning for his next trip to the unknown corners of the world.

Sticker Pack Apps

Sticker Pack Apps can be created in less than 30 seconds because they're code free, meaning that they can be developed and published without writing a single line of code. Xcode 8 provides a new template for creating Sticker Pack Applications, as shown in **Figure 1**.

Once you select the **Sticker Pack Application** template, Xcode 8 automatically creates the default starter project. An important thing to notice is that the Sticker Pack Application project doesn't contain storyboard, view controllers, or even application delegate files. That's because Sticker Pack Applications are 100% code-free.

Select the **Stickers.xcstickers** assets and in the middle pane, select the **Sticker Pack** folder. This updates the right pane, which displays the number of stickers in the application. Because you haven't added any stickers to the application yet, it says **No Stickers—Drag and drop images to add new stickers.**

At this point, all you have to do is drag and drop the images into the designated area and you're done. I told you: This is going to take less than 30 seconds!

Figure 3 shows the Sticker Pack Application after I've dropped several awesome smiley face emojis into the sticker assets folder.

At this point, you're done with the app! You can run the app and witness your awesome work.

By default, the Sticker Pack Applications displays the stickers in medium sticker-size format. You can adjust the size using the attributes inspector, as shown in **Figure 5.**

If you run the app again, you'll notice that the stickers are much bigger in size. For the larger size stickers, make sure that your stickers' dimensions are greater than 500 x 500 points. The stickers with large sizes create a very distinctive and unique effect, as shown in **Figure 6**.

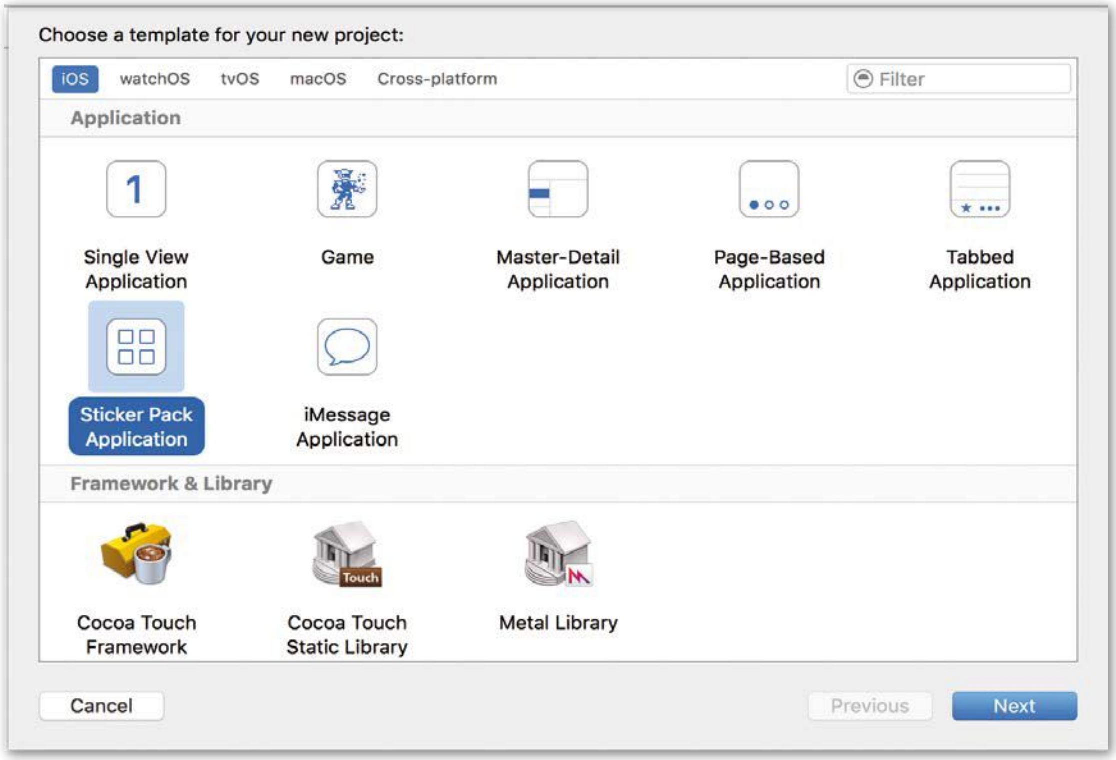
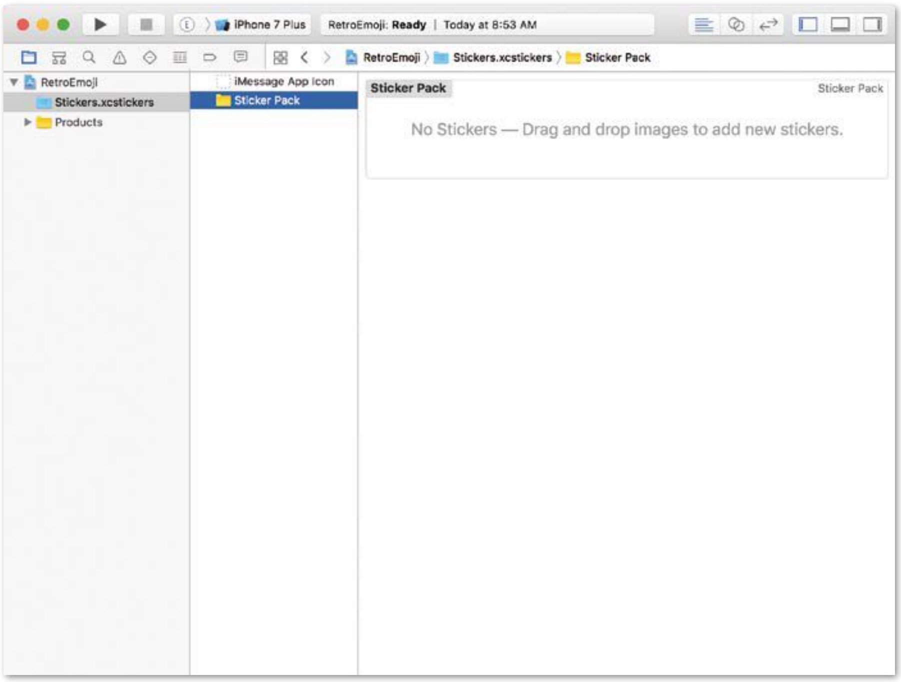


Figure 1: Sticker Pack Application Template in Xcode 8

Even though sticker applications are quick and easy to create and are great for sharing funny emojis, pictures, GIF images, etc., they're quite limited in nature. The limitation comes from the fact that sticker applications don't have any code. This means that sticker pack applications can't integrate with ad networks, in-app purchases or even transition to different screens.

Sticker Pack applications allow the designers to use their creative skills to step into the arena of app development without having to write a single line of code.

Don't feel upset that you can create the same sticker application using code. Instead of choosing the sticker pack application project template, you use the iMessages application project template. In the next section, I'll demonstrate how you can write a complete iMessages application that allows the user to keep track of RSVP confirmations.



iPhone 7 Plus
RetroEmoji: Ready | Today at 8:53 AM
RetroEmoji
Stickers.xcstickers
Sticker Pack
Products
iMessage App Icon
No Stickers — Drag and drop images to add new stickers.

Figure 2: Xcode 8 Stickers Pane Area for Sticker Pack applications

Creating the RSVP iMessages Application

The real power of the new iOS 10 Messages framework comes from the iMessages applications. Just like sticker pack applications, iMessages applications run under the umbrella of the Messages framework app but provide a lot more flexibility for customization.

In this section, I'm going to create a very practical iMessages application for managing RSVPs. Consider a scenario where you're throwing a birthday party and you want an RSVP count for the number of guests. Instead of sending and tracking hundreds of RSVPs on your Messages app, wouldn't it be great if you could simply get the count of people who're interested in attending the event? This saves you from the headache of counting the confirmations yourself and also affects the number of pizzas you order for the party. Nobody likes to waste pizza and cake, am I right?

Messages App Presentation Styles

Before jumping into the technical details of the RSVP iMessages app, I'll talk about the different presentation styles of the iMessages application. The iMessages app comes with two presentation styles: compact and expanded.

- **Compact:** The user selects the extension in the app drawer: The Messages app launches the extension using the compact style.
- **Expanded:** The user selects a message in the transcript that represents one of the extension's MSMessage objects: The Messages app launches the extension using the expanded style.

Adding Child View Controllers

When creating iMessages applications, Xcode automatically adds a default view controller on the storyboard.

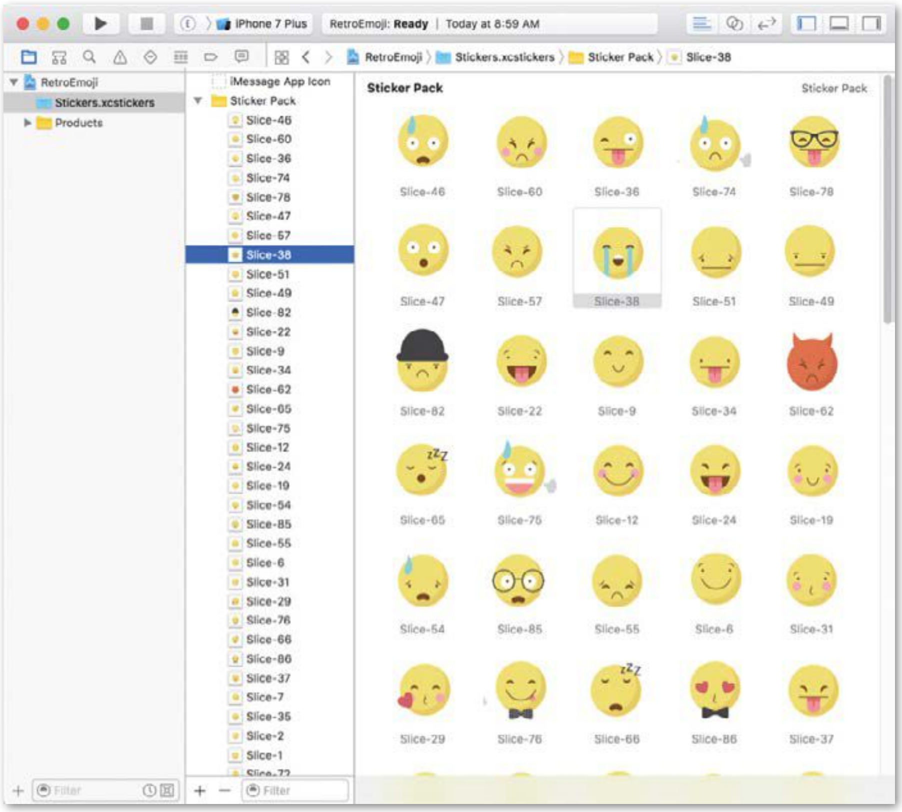


Figure 3: Displaying a list of emojis as part of the Sticker Pack App

This view controller inherits from MSMessagesAppViewController and serves as the root controller for the iMessages applications. In order to display a custom view controller, you need to inject the controller inside the MSMessagesAppViewController using view controller containment.

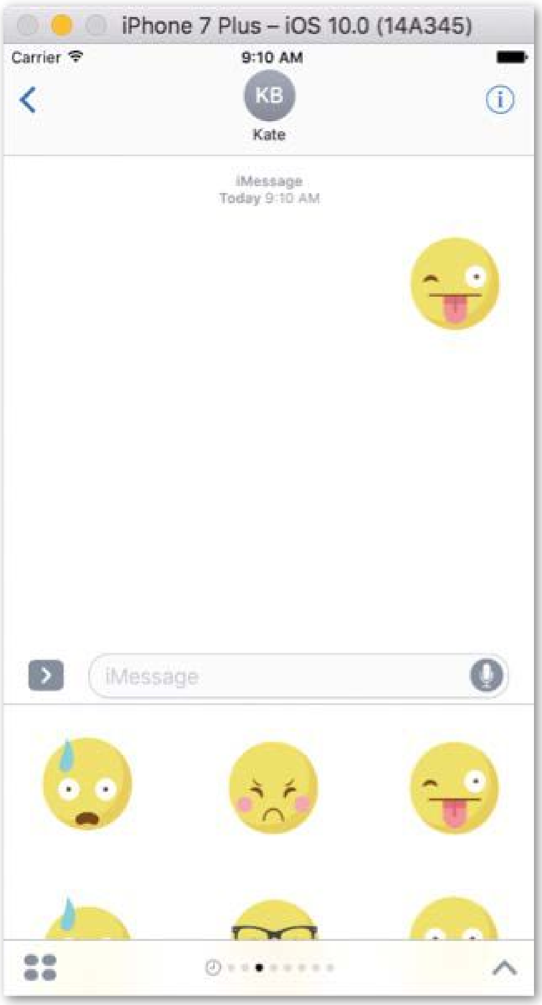


Figure 4: Sticker Pack App running in the iPhone simulator

Because the RSVP app requires several screens, it's a good idea to separate the functionality of each screen into a separate view controller. This technique allows you to follow the single responsibility principles and helps you write more maintainable and readable code. **Figure 7** shows the overall user interface architecture of the app.

You can see that:

- The **CompactViewController** is displayed in the compact mode when the application is launched.
- The **CreateRSVPViewController** allows the user to create a new RSVP and enter the details of the event.
- The **SelectionViewController** allows the user to respond to the RSVP request.

The first step is to add the CompactViewController to the MessagesViewController using the view controller containment techniques. I've created a custom function that's responsible for adding and removing child view controllers. **Listing 1** shows the complete implementation of the presentViewController function.

Depending on the presentation style, the **presentViewController** function injects the correct controller into the MessagesViewController. The **instantiateRSVPCompactViewController** and **instantiateCreateRSVPViewController** functions are responsible for instantiating the controllers based on their storyboard Identifiers. **Listing 2** shows the instantiate functions implementation.

Run the app and you'll notice that the RSVPCompactViewController is injected into the MessagesViewCon-

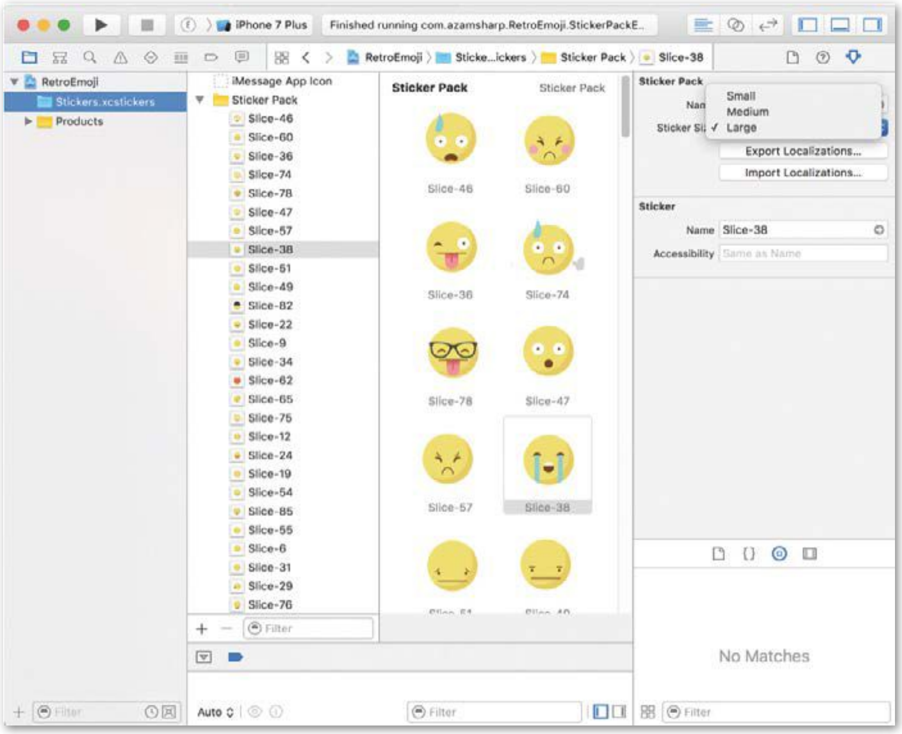
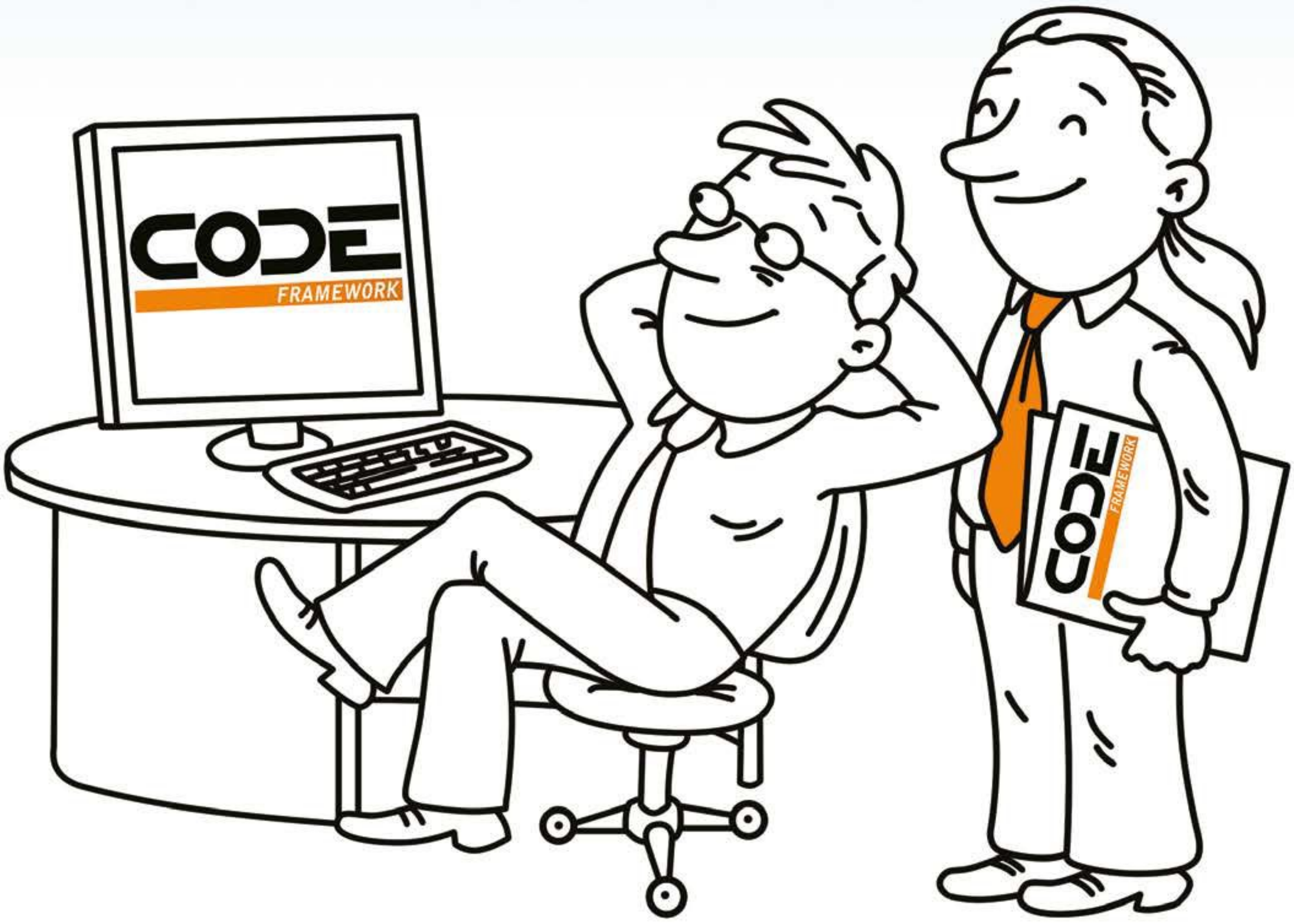


Figure 5: Adjusting the size of the stickers icons

Figure 6: Emoji King App with enormously huge emojis.

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

Listing 1: A generic function to inject designated view controller into the MSMessagesAppViewController

```
private func presentViewController
(with conversation:MSConversation
, for presentationStyle :
MSMessagesAppPresentationStyle) {

var controller :UIViewController!

if presentationStyle == .compact {

controller = instantiateRSVPCompactViewController()

} else {


controller = instantiateCreateRSVPViewController()


}

//Remove any existing child controllers.
for child in childViewControllers {
child.willMove(toParentViewController: nil)
child.view.removeFromSuperview()
child.removeFromParentViewController()
}

addChildViewController(controller)

controller.view.frame = self.view.bounds
controller.view.
translatesAutoresizingMaskIntoConstraints
 = false
    self.view.addSubview(controller.view)

controller.view.leftAnchor.constraint(equalTo:
view.leftAnchor).isActive = true
    controller.view.rightAnchor.constraint(equalTo:
view.rightAnchor).isActive = true
    controller.view.topAnchor.constraint(equalTo:
view.topAnchor).isActive = true
    controller.view.bottomAnchor.constraint(equalTo:
view.bottomAnchor).isActive = true

controller.didMove(toParentViewController: self)
}
```

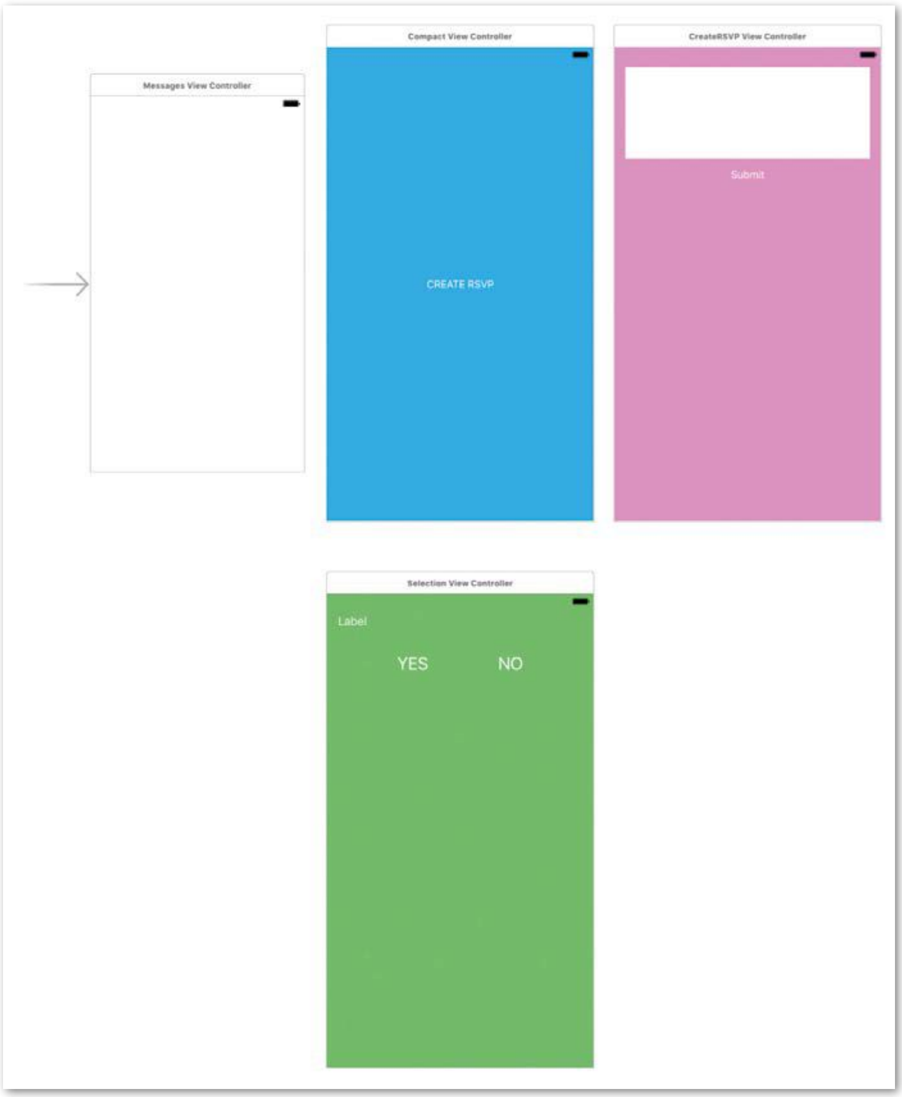


Figure 7: User Interface Layout of the iMessages RSVP Application

troller and is displayed in the compact view due to its presentation style. If you press the expand button, the presentViewController is triggered again and CreateRSVPViewController is injected into the MessagesViewController. **Figure 3** shows this in action.

The communication between the child view controllers and the parent view controller is performed using protocols and delegates. The CompactRSVPViewController exposes a protocol that can be implemented by the delegate classes. The implementation of the instantiateRSVPCompactViewController function already set its delegates, and are notified when any protocol function is triggered. The protocol CompactRSVPViewController delegate defines only a single function called rsvpCompactViewControllercreateRSVP. The function doesn't take any parameters and it doesn't return any parameters. The only purpose of this delegate method is to transfer the control back to the MessagesViewController, as can be seen in the following snippet.

Luckily, the MSMessages class allows you to send customized messages

```
protocol RSVPCompactViewControllerDelegate :
class {

    func rsvpCompactViewControllercreateRSVP()

}
```

The MessagesViewController conforms to the RSVPCompactViewControllerDelegate and provides a custom implementation for the rsvpCompactViewControllercreateRSVP function, as shown in the next snippet.

```
func rsvpCompactViewControllercreateRSVP() {

    requestPresentationStyle(.expanded)
 }
```

As you can see, the rsvpCompactViewControllercreateRSVP simply requests the expanded presentation style, which launches the CreateRSVPViewController. The requestPresentationStyle function invokes the didTransition function, which in turn calls the presentViewController function and selects the correct controller depending on the passed presentation style.

Customizing Messages

Once the user fills in the RSVP description, you need to communicate that to the other recipient in the form of a message. The Messages Framework provides multiple ways of sending messages. A message can be sent as a plain text message or a customizable MSMessage. The next snippet shows an approach where a plain text message is sent as an RSVP invitation.

```
func createRSVPViewControllerDidCreatedRSVP
(rsvp: RSVP) {
```

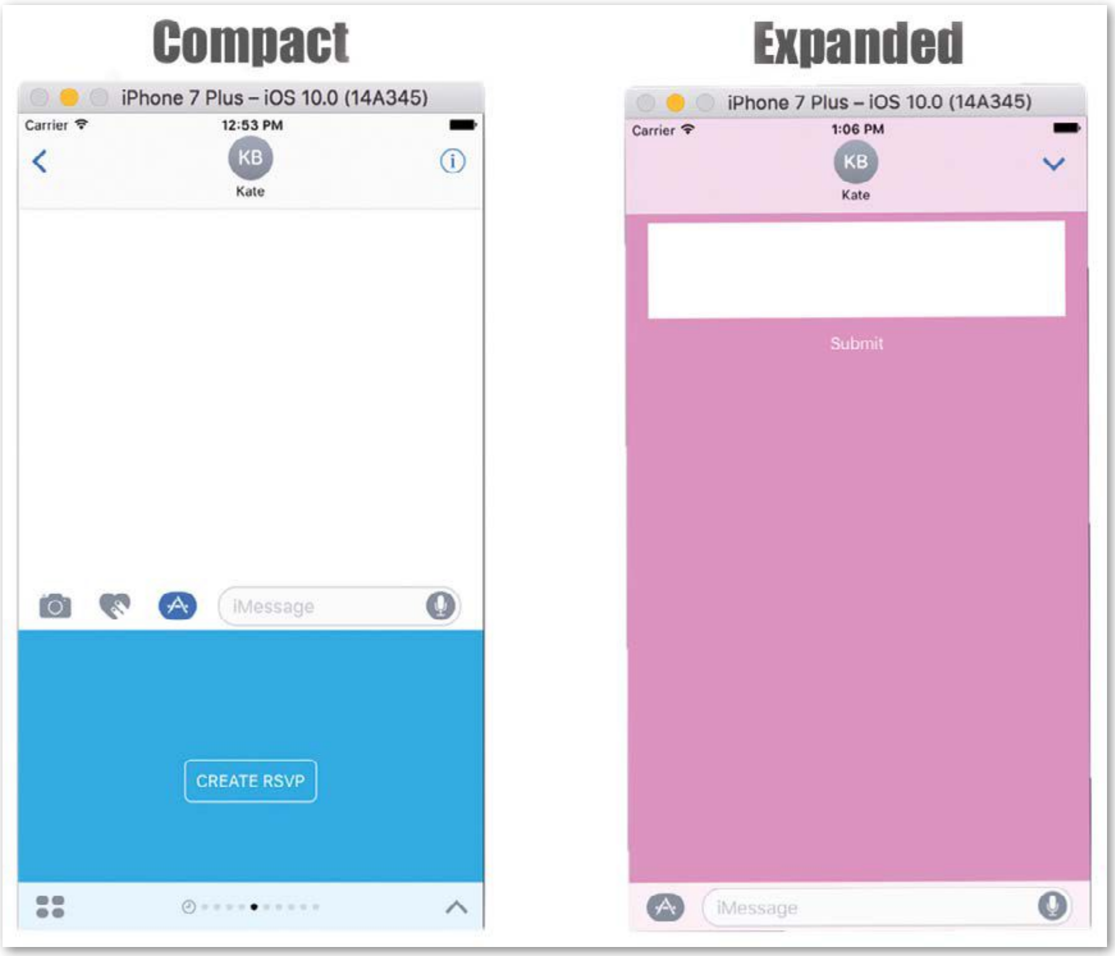


Figure 8: iMessages compact and expanded states

Listing 2: Instantiating the Child Controllers

```
private func
 instantiateCreateRSVPViewController()
 -> UIViewController {

guard let controller = self.storyboard?
.instantiateViewController
(withIdentifier: "CreateRSVPViewController")
 as? CreateRSVPViewController else {
fatalError("CreateRSVPViewController not found")
}

controller.delegate = self
return controller
}
```

```
private func instantiateRSVPCompactViewController()
 -> UIViewController {

guard let controller = self.storyboard?
.instantiateViewController
(withIdentifier: "RSVPCompactViewController") as?
 RSVPCompactViewController else {
fatalError("RSVPCompactViewController not found")
}

controller.delegate = self
return controller
}
```

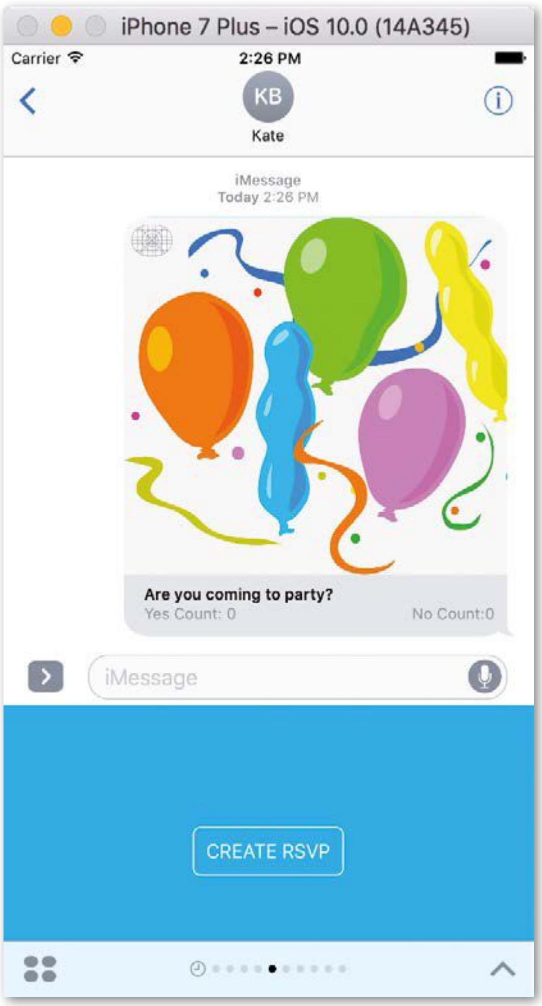


Figure 9: Use the iMessages App to send a simple text message.

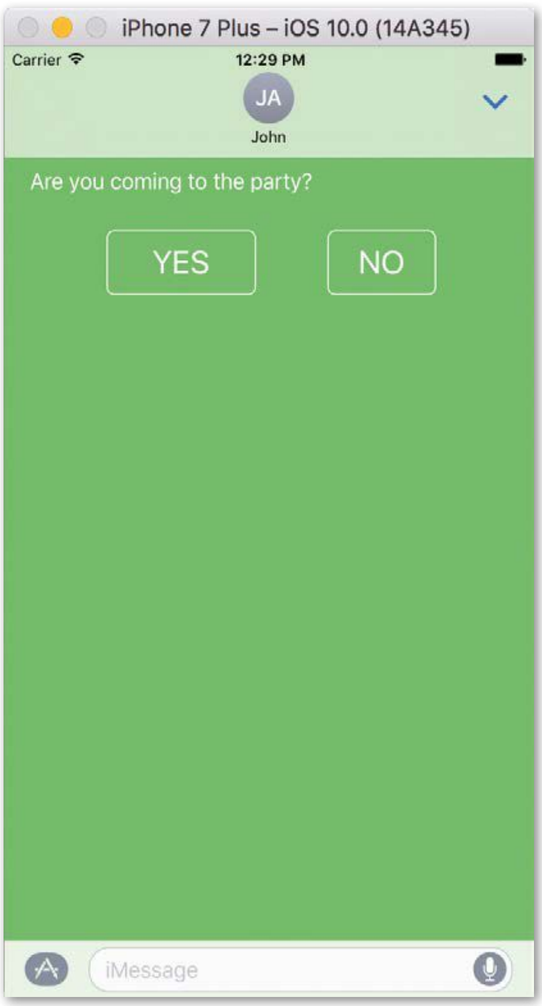

iPhone 7 Plus – iOS 10.0 (14A345)
Carrier
12:29 PM
JA
John
Are you coming to the party?
YES
NO
iMessage

Figure 10: Message custom layout

Listing 3: Function to create a message with custom layout

```
private func composeMessage(rsvp :RSVP) ->
 MSMessage {

let message = MSMessage()
let components = NSURLComponents()
message.url = components.url!

let layout = MSMessageTemplateLayout()
layout.image = UIImage(named: "partyballoons.jpg")
layout.caption = rsvp.title
layout.subcaption = "Yes Count: \(rsvp.yes)"
layout.trailingSubcaption = "No Count:\(rsvp.no)"

message.layout = layout
return message
}
```

```
  requestPresentationStyle(.compact)

  self.activeConversation?.insertText(rsvp.title,
completionHandler: nil)

}
```

The result is shown in **Figure 9.**

It's pretty simple! But it also doesn't capture any enthusiasm and excitement for the party. Luckily, the MSMessages class allows you to send customized messages. **Listing 3** shows the implementation of the composeMessage function, which creates a MSMessage class.

Apple simulator doesn't provide the functionality to send group messages nor does it allow you to add custom Messages accounts.

The composeMessage function takes in an object of type RSVP class and generates and returns an instance of the MSMessage class. The MSMessageTemplateLayout is responsible for customizing the look and feel of the message. I assigned an image to the layout that serves as the background image of the message. I also changed the caption of the message through the template layout, which allowed me to show the text of the invitation. **Figure 10** shows the result in action.

This definitely looks much better, and I'd bet that more people are going to RSVP **YES** for the party due to the appealing appearance of the message.

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

Listing 4: Compose Function with Sharing Data Through URL

```
private func composeMessage(rsvp :RSVP)
 -> MSMessage {

let session = self.activeConversation?
.selectedMessage?.session

let message = MSMessage(session: session
 ?? MSSession())

let components = NSURLComponents()

components.queryItems = [URLQueryItem
(name: "title", value:
rsvp.title),URLQueryItem
(name:"yesCount",value:"\(rsvp.yes)"),
URLQueryItem(name:"noCount",value:"\(rsvp.no)")]

message.url = components.url!

let layout = MSMessageTemplateLayout()
layout.image = UIImage(named: "partyballoons.jpg")
layout.caption = rsvp.title
layout.subcaption = "Yes Count: \(rsvp.yes)"
layout.trailingSubcaption = "No Count:\(rsvp.no)"

message.layout = layout

return message

}
```

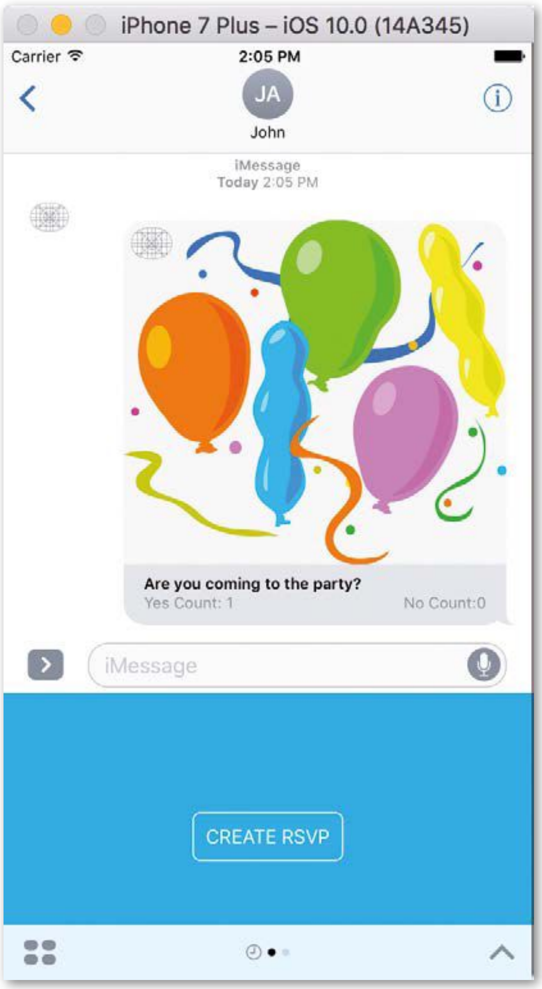


Figure 11: RSVP confirmation screen

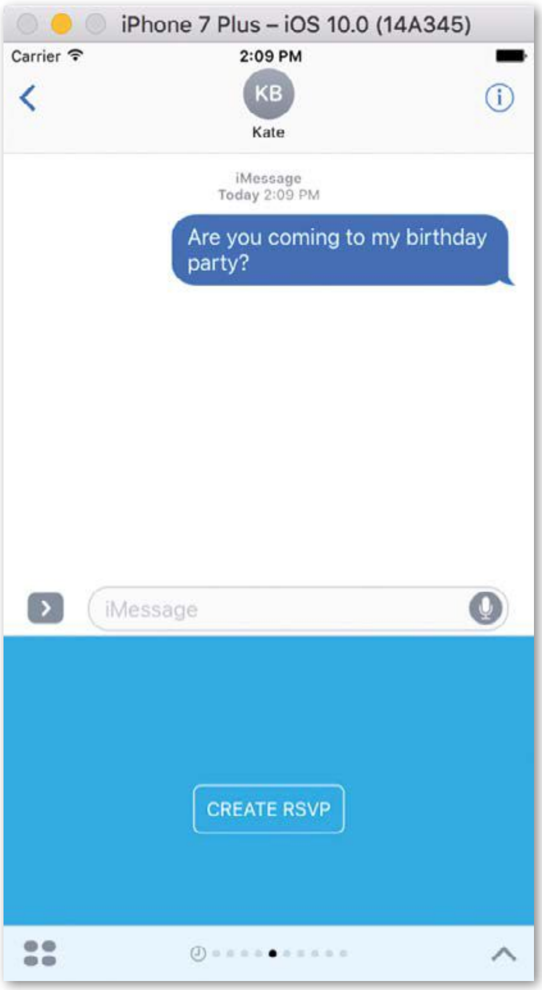

iPhone 7 Plus – iOS 10.0 (14A345)
Carrier
2:09 PM
KB
Kate
iMessage
Today 2:09 PM
Are you coming to my birthday party?
iMessage
CREATE RSVP

Figure 12: Message updated with RSVP count

Maintaining Sessions Between Participants

In order to update the RSVP count, you need to maintain the session between the participants. This can be accomplished by using the URL property of the MSMessage class. The URL property can hold query string data that can represent the information in pairs. The next snippet shows the query string to create for the application.

```
message=Are%20you%20coming%20to%20my%20
party&yesCount=54&noCount=3
```

The query string contains three pieces of information in pairs. Each pair is identified by a key and a value. Here's the breakdown of the message URL:

- **message:** The actual text of the invitation
- **yesCount:** The number of people who replied yes
- **noCount:** The number of people who replied no

Listing 4 shows the implementation of the composeMessage function, which adds the query parameters to the message.

The NSURLComponents class is responsible for creating the query string in a designated format. Finally, you assign the newly generated URL to the message's URL property. Now when a message is added to the active conversation, it has the underlying query string part of the URL property. Next, you need to add a view that displays the text of the invitation and also to accept or reject the invitation.

You can't modify the original message, but you can append to it by using a shared session between participants.

The RSVPSelectionViewController is responsible for providing the user with options to either accept or reject the invitation. The RSVPSelectionViewController consists of two buttons, one for YES and the other one for NO, as shown in **Figure 11**.

The RSVP title is passed to the RSVPSelectionViewController using the message's URL property, where it can be decomposed into an object and then displayed in the view controller. Finally, the user is given the opportunity to accept or reject the invitation by pressing a button. The button, when pressed, increments the yes or no count and then appends a new message to the active conversation. The result is shown in **Figure 12**.

Conclusion

Apple's inclusion of the Messages framework has opened doors for creating a new category of applications. Apart from the stickers apps, iMessages apps have become an integral part of Apple's echo system, allowing you to create a new form of communication medium for your users. The Messages framework has allowed developers to extend the existing iOS app to include the messages capability.

Mohammad Azam

SPONSORED SIDEBAR:

Ramp up with CODE

Are you stuck at project roadblocks? CODE Training offers a popular 20-Hour Introductory Mentoring Service to help you get moving again. You get 20 hours of full access to a CODE team member for help on any development project, which is time that can be used for one-on-one mentoring or to give your project the push it needs to get started. To find out more about this offer, email info@codemag.com and set up your time with a CODE specialist today!

Accessing Your Data with F# Type Providers

In the last issue (January/February 2017), I covered how easy it is to work with F# type providers specifically for data science. This time, I'll back up a little and show you just how easy it is to simply access (and still work with!) most data that you'll come across. You'll see examples for accessing XML, JSON, and APIs that use the new Swagger type provider. Next issue, I'll write an entire article covering the options for SQL Server, so get ready!

Rachel Reese

rachelree.se
twitter.com/rachelreese

Rachel Reese is a long-time software engineer and math geek who can often be found talking to random strangers about the joys of functional programming and F#. She holds a Bachelor's of Science in Math and Physics from Stony Brook University, and has a habit of starting user groups: so far, in Hoboken, NJ (Pluralsight Study Group), Nashville, TN (@NashFSharp) and Burlington, VT (@VTFun). She's also on the F# Software Foundation board, the .NET Foundation board, and is an ASPInsider, an F# MVP, a Xamarin MVP, a community enthusiast, one of the founding @lambdaladies, and a Rachii. You can find her on twitter, @rachelreese, or on her blog: rachelree.se.

XML Type Provider

Working with XML is super easy when you have the XML type provider on hand. I happen to have a data set that shows the Powerball winning lottery numbers for the State of New York Lottery from February 2010 through January 2017. Let's see how the most common numbers chosen in each month differ. First, because it's a script file, you'll need to include your references and open statements. To use the XML type provider, you'll need **FSharp.Data**, as well as **System.Xml.Linq**, because the type provider makes use of **XDocument** internally. I'm also using **Microsoft.VisualBasic**, because it has a few nice **Date** functions that the other languages don't have.

```
#r "../packages/FSharp.Data/lib/FSharp.Data.dll"
#r "System.Xml.Linq.dll"
#r "Microsoft.VisualBasic.dll"
open FSharp.Data
open Microsoft.VisualBasic
```

Once these are in place, you connect to the data. It's a quick two lines: you simply tell the type provider where the data can be found and then call **GetSample()**. You could also have used a document that's an example of the data and then used the **Parse** method on the full data.

```
type Names = XmlProvider<"../numbers.xml">

let names = Names.GetSample()
```

That's it! You're ready to use the data, which has the following form:

```
<row _id="931"
  _uuid="ABE7162C-2AB8-4CF1-B7DD-00D473BC7960"
  _position="931"
  _address="http://data.ny.gov/resource/931">
    <draw_date>2017-01-04T00:00:00</draw_date>
    <winning_numbers>
      16 17 29 41 42 04
    </winning_numbers>
    <multiplier>3</multiplier>
</row>
```

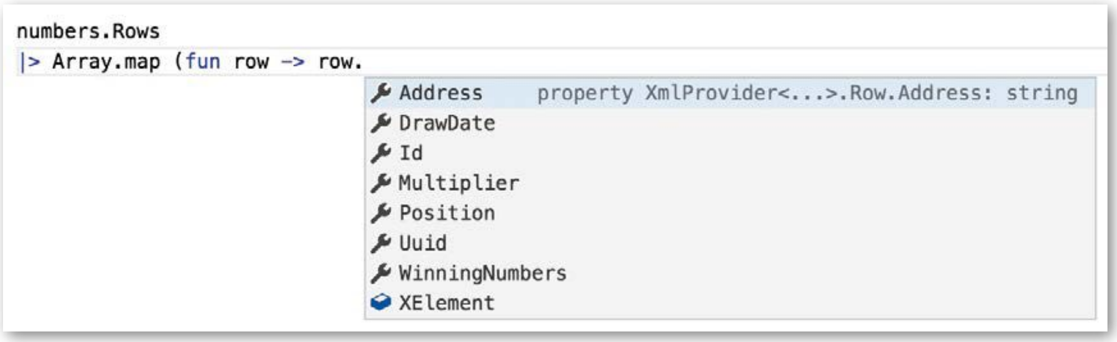
Now, check out **Figure 1**, and note how the type provider gives you that data.

It's not clear from **Figure 1** or from the data, but the winning numbers are returned all together as a string. Before you can do much with them, you'll need to parse them into an array of numbers. This is easily done by splitting the string and converting each result to an integer, like so:

```
let parseNumbers (nums:string) =
  nums.Split(' ')
  |> Array.map (fun n -> n |> int)
```

Now that you've created the **parseNumbers** function, it's time to start forming your results. Let's call this **results**. You can request all the rows of the data using **names.Rows**, and then use an **Array.map** to select only the data you're looking for from the XML file: the month that the numbers were drawn and the numbers themselves parsed into an array. Once that's in place, you can group by the month and then sort so they're in the correct order.

```
let results =
  names.Rows
  |> Array.map
    (fun r ->
```



```
numbers.Rows
|> Array.map (fun row -> row.
                          Address      property XmlProvider<...>.Row.Address: string
                          DrawDate
                          Id
                          Multiplier
                          Position
                          Uuid
                          WinningNumbers
                          XElement
```

Figure 1: Using IntelliSense to explore the NY State lottery numbers XML data file

```
      (r.DrawDate.Month,
      parseNumbers r.WinningNumbers))
  |> Array.groupBy (fun (m, n) -> m)
  |> Array.sort
```

This gives you a result like **Listing 1**.

However, now you need to clean up and combine all the sub-arrays so that you can more meaningfully interact with your data. Let's create a **combineNumbers** function to handle them. I'll walk through it line-by-line for some clarity.

```
let combineNumbers data =
  data
  |> Array.collect
    (fun (month, nums) -> nums)
  |> Array.countBy id
  |> Array.sortByDescending
    (fun (num, count) -> count)
  |> Array.map (fun (n,c) -> n)
```

I started by sending the data to an **Array.collect**. This runs the function on each item in the array and then collects and concatenates the resulting sub-arrays. This removes the extra month information and gathers all the lottery numbers into one large array.

```
  data
  |> Array.collect
    (fun (month, nums) -> nums)
```

Next, I count the numbers. Because there's only one term per item in the area, there's no need to use an explicit **countBy** function; **id** will suffice. The result here is an array of tuples: the count and the lottery number.

```
  |> Array.countBy id
```

Next, I sort by the count to return the highest chosen numbers in that month.

```
  |> Array.sortByDescending
    (fun (num, count) -> count)
```

Finally, I remove the count and return only the numbers.

```
  |> Array.map (fun (n,c) -> n)
```

Now I can take the **combineNumbers** function and add another step to the **results** computation. In this last step, I'll map the month and numbers to a proper month name string, and I'll use the **combineNumbers** function to clean up the sub arrays and take just the top three lottery numbers that were returned.

```
let results =
  names.Rows
  |> Array.map
    (fun r ->
      (r.DrawDate.Month,
      parseNumbers r.WinningNumbers))
  |> Array.groupBy (fun (m, n) -> m)
  |> Array.sort
  |> Array.map
    (fun (m, data) ->
```

Listing 1: Results of lottery data after parsing numbers, grouping by month, and sorting.

```
val results : (int * (int * int []) []) [] =
  [|
    (1,
      [|(1, [|18; 22; 37; 47; 54; 36|]);
        (1, [|22; 26; 32; 38; 40; 7|]);
        ...
      |]);
    (2,
      [|(2, [|17; 22; 36; 37; 52; 24|]);
        (2, [|14; 22; 52; 54; 59; 4|]);
        ...
      |]);
    (3,
      [|(3, [|7; 9; 14; 45; 49; 23|]);
        (3, [|10; 29; 33; 41; 59; 15|]);
        ...
      |]);
      ...
    ...
  |]
```

```
      (DateAndTime.MonthName(m),
      (combineNumbers data).[0..2]))
```

The final result looks like this!

```
val results : (string * int []) [] =
  [|
    ("January", [|47; 19; 28|]);
    ("February", [|17; 11; 36|]);
    ("March", [|14; 8; 12|]);
    ("April", [|33; 29; 39|]);
    ("May", [|23; 31; 9|]);
    ("June", [|22; 33; 18|]);
    ("July", [|3; 11; 38|]);
    ("August", [|12; 28; 4|]);
    ("September", [|17; 22; 39|]);
    ("October", [|20; 10; 1|]);
    ("November", [|17; 37; 5|]);
    ("December", [|10; 14; 19|])
  |]
```

JSON Type Provider

The JSON type provider is similar to the XML type provider. Here, too, you can point the type provider to a full data set, an example data set, or a URL that returns the results in JSON. This time, I'll show an example of making a call to an API that returns JSON. I'll use the Broadband Speed Test API to get some state averages for download speeds for a few different places (libraries, home, mobile, etc.) and then compare those to the national averages. Finally, I'll chart the numbers.

You've always got to start with your references and open statements. In this case, you need to load the **FsLab** scripts, and reference the **GoogleCharts** dll. You also need to open the **GoogleCharts** library.

```
#load "../packages/FsLab/FsLab.fsx"
#r "XPlot.GoogleCharts.dll"

open XPlot.GoogleCharts
```

Next, you can set up the two type providers. You'll need two separate instances because you're making two separate calls. I've chosen to call mine **BroadbandStates** and **BroadbandNation**. Then you call **GetSample**(), just like the XML type provider, and you're ready to go.

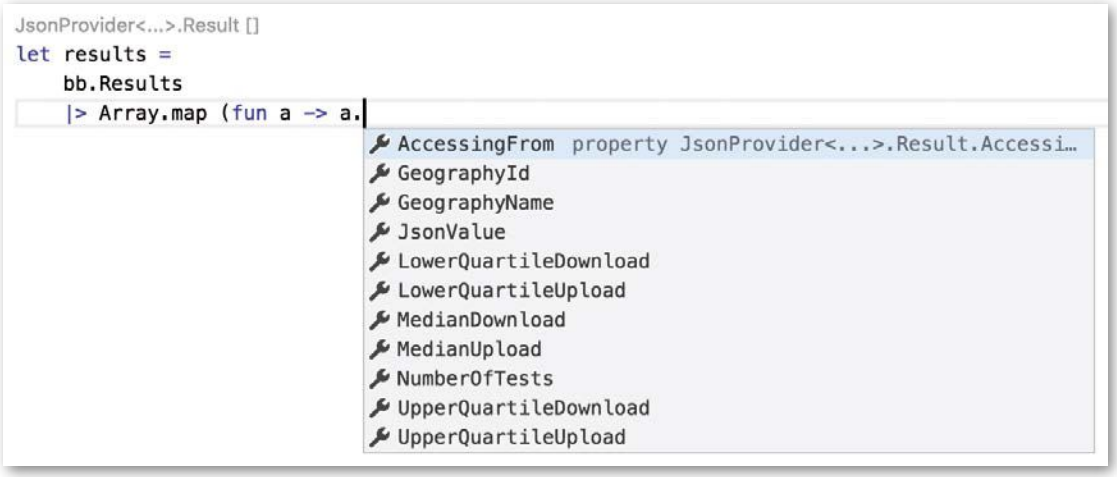
```
let statesUrl =
  "https://www.broadbandmap.gov/broadbandmap/" +
  "speedtest/state/names/arizona,maine," +
  "wyoming,california,tennessee?format=json"

let nationUrl =
  "https://www.broadbandmap.gov/broadbandmap/" +
  "speedtest/nation?format=json"

type BroadbandStates = JsonProvider<statesUrl>
type BroadbandNation = JsonProvider<nationUrl>

let bbStates = BroadbandStates.GetSample()
let bbNation = BroadbandNation.GetSample()
```

Working with the national data is a little easier, so let's start there. You'll need to access the results, and then use **Array.map** to find the two points of data that you need: **AccessingFrom**, and **MedianDownload**. Check out **Figure 2** to see all the options available.

```
let nationResults =
  bbNation.Results
  |> Array.map
    (fun a -> (a.AccessingFrom, a.MedianDownload))
```

Next, you'll set up the state results in a similar manner, as in the next snippet. This time, you'll need to return the **GeographyName** as well as the **AccessingFrom** and **MedianDownload**, and then, because the goal is to find the percent difference in the average download speed as compared to the national average download speed, you'll want to group by state.

```
  bbStates.Results
  |> Array.map
    (fun a ->
    (a.GeographyName,
     a.AccessingFrom,
     a.MedianDownload))
```



```
JsonProvider<...>.Result []
let results =
    bb.Results
    |> Array.map (fun a -> a.
```

AccessingFrom property JsonProvider<...>.Result.Accessi…
GeographyId
GeographyName
JsonValue
LowerQuartileDownload
LowerQuartileUpload
MedianDownload
MedianUpload
NumberOfTests
UpperQuartileDownload
UpperQuartileUpload

Figure 2: Using IntelliSense to explore the Broadband Speed Test API

Listing 2: NationResults and StateResults data. Note that each set of state information needs to line up with the national information in order for the Array.map2 function to work properly.

```
val nationResults : (string * decimal) [] =
  [|("Business", 8.85742M);
    ("CC_library_school", 9.98828M);
    ("Home", 6.70215M); ("Mobile", 2.12891M);
    ("Other", 3.97627M); ("Small_Business", 4.39063M)|]

val stateResults : (string*(string*string*decimal)[])[] =
  [|("Arizona",
     [|("Arizona", "Business", 8.73516M);
       ("Arizona", "CC_library_school", 9.66016M);
       ("Arizona", "Home", 7.45703M);
       ("Arizona", "Mobile", 2.18652M);
       ("Arizona", "Other", 6.51075M);
       ("Arizona", "Small_Business", 4.30176M)|]);
    ("California",
     [|("California", "Business", 8.85303M);
       ("California", "CC_library_school", 18.33496M);
       ("California", "Home", 6.13222M);
       ("California", "Mobile", 2.58984M);
       ("California", "Other", 2.51492M);
       ("California", "Small_Business", 3.19922M)|]);
    ("Maine",
     [|("Maine", "Business", 10.89160M);
       ("Maine", "CC_library_school", 8.88086M);
       ("Maine", "Home", 5.34131M);
       ("Maine", "Mobile", 1.89844M);
       ("Maine", "Other", 4.80187M);
       ("Maine", "Small_Business", 4.35449M)|]);
    ("Tennessee",
     [|("Tennessee", "Business", 7.91162M);
       ("Tennessee", "CC_library_school", 8.63184M);
       ("Tennessee", "Home", 7.31219M);
       ("Tennessee", "Mobile", 2.42871M);
       ("Tennessee", "Other", 1.72534M);
       ("Tennessee", "Small_Business", 4.95801M)|]);
    ("Wyoming",
     [|("Wyoming", "Business", 4.32042M);
       ("Wyoming", "CC_library_school", 6.40321M);
       ("Wyoming", "Home", 3.86426M);
       ("Wyoming", "Mobile", 1.20020M);
       ("Wyoming", "Other", 2.28906M);
       ("Wyoming", "Small_Business", 2.59903M)|])|]
  |> Array.groupBy
    (fun (state, from, down) -> state)
```

After this code runs, the **stateResults** and **nationResults** values look like **Listing 2**.

Next, you need a helper function to handle calculating the averages for each state. I've called it **findAverages**. Let's look a little closer at that one. I take the data and use an **Array.map2**, which takes two arrays as inputs and performs a map on them, using their respective elements. For example, generically, Array.map2 might look like this:

```
Array.map2
  (fun array1value array2value ->
    array1value + array2value)
    array1 array2
```

Or, of course, like this:

```
array2
|> Array.map2
    (fun array1value array2value ->
      array1value + array2value)
      array1
```

The arrays must be the same length and conveniently, now each state sub-array is the same length as the array of national data. It's also useful here to pre-emptively destructure the tuple and triple in each array. Your **findAverages** function will look like this:

```
let findAverages data =
  data
  |> Array.map2
    (fun (nfrom, ndown) (geo, sfrom, sdown) ->
      (geo, sfrom, sdown/ndown*100M |> int))
      nationResults
```

Once you have the averages, you want to re-group by the location (home, office, etc.) so that you can chart the data, and then remove the duplicated location information. Now, let's put this together.

```
  bbStates.Results
  |> Array.map
    (fun a ->
    (a.GeographyName,
     a.AccessingFrom,
     a.MedianDownload))
  |> Array.groupBy
    (fun (geo, from, down) -> geo)
  |> Array.collect
    (fun (geo, data) -> findAverages data)
  |> Array.groupBy
    (fun (geo, from, down) -> from)
  |> Array.map
    (fun (from, data) -> (from, sortData data))
```

In that snippet, sortData is the function that removes the duplicated location information.

```
let sortData (data:(string*string*int)[]) =
  data
  |> Array.map
    (fun (geo, from, down) -> geo, down)
```

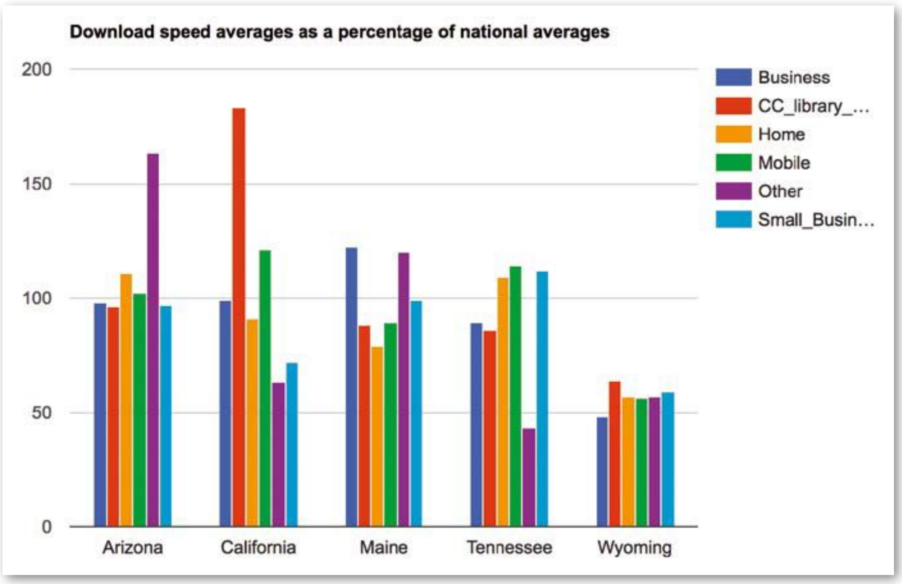


Figure 3: Chart of the values from the Broadband Speed Test API for Arizona, California, Maine, Tennessee, and Wyoming download speeds in various locations

There's one last step to process the state data. You'll have an array of tuples here: a string value (the location information), and an array of tuples that are a string (the state) and an integer (percent difference to national value). Let's unzip this array, using the location information for the labels on the chart, and use compound assignment to set both values at once. You have, finally:

```
let labels, stateResults =
  bbStates.Results
  |> Array.map
    (fun a ->
    (a.GeographyName,
     a.AccessingFrom,
     a.MedianDownload))
  |> Array.groupBy
    (fun (geo, from, down) -> geo)
  |> Array.collect
    (fun (geo, data) -> findAverages data)
  |> Array.groupBy
    (fun (geo, from, down) -> from)
  |> Array.map
    (fun (from, data) -> (from, sortData data))
  |> Array.unzip
```

Now, to create a chart of all the data at once, you need to specify which type of chart each group of data should be. In this case, you'll want all bar charts.

```
let series =
  ["bars"; "bars"; "bars"; "bars"; "bars"; "bars"]
```

Finally, let's chart those results. I used a combo chart, added a title, and used the series information to set each chart up as a bar chart. I added the labels information and a legend. Voila! See **Figure 3** for the final chart.

```
stateResults
|> Chart.Combo
|> Chart.WithOptions
  (Options(
```

The WSDL Type Provider

When I give talks on type providers, I often include a demo of the WSDL type provider. WSDL obviously isn't new and cool any more, but it was the first type provider that I was able to see in action and no article on data access would be properly complete without at least giving it a mention. For more information and documentation on this type provider, see https://msdn.microsoft.com/en-us/visualfsharpdocs/conceptual/wsdlservice-type-provider-%5Bfsharp%5D

```
    title = "Download speed averages as a
      percentage of national averages",
    series = [| for typ in series ->
      Series(typ) |]))
|> Chart.WithLabels labels
|> Chart.WithLegend true
```

The JSON type provider is fantastic, but it does mean that you need to set up each individual call separately. To solve this problem, let's take a look at the Swagger type provider.

Swagger Type Provider

First, what is Swagger? It's a formal specification that ensuring that your API is documented in a human-readable and a machine-readable way, with minimal fuss. Using Swagger for your APIs is the new hotness right now, and this means you can take advantage of the several APIs that use Swagger, via the Swagger type provider. You'll have quick, full access to the entire API!

Using Swagger for your APIs is the new hotness right now.

I've written up a short script that uses the New York Times API to read their top headlines for the Travel section, and then does a bit of processing on the words in the article abstracts, using the Oxford dictionary API, in order to determine how many words are used. Let's take a look!

Much the same as in the previous examples, you need to specifically call out your references and open statements. You need to reference the **SwaggerProvider**, which can be loaded with either Paket or Nuget. I also make a habit of keeping my API keys in a separate file so that I don't accidentally make them public. For this bit of code, you need to reference **System.Text.RegularExpressions** and **System.Threading**.

```
#load "../packages/Swagger/SwaggerProvider.fsx"
#load "keys.fs"

open SwaggerProvider
open keys
open System.Text.RegularExpressions
open System.Threading
```

Next, let's connect to the Oxford dictionary API. Using the **SwaggerProvider**, it's a matter of pointing the type provider at the Swagger definitions for the API and creating an instance for use.

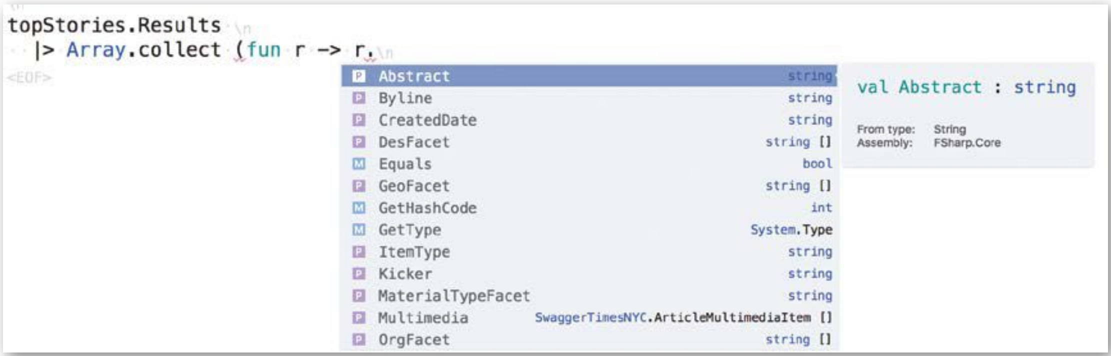
```
[<Literal>]let oxfordApi =
 «https://api.apis.guru/v2/specs/» +
    "oxforddictionaries.com/1.4.0/swagger.json"
type Oxford = SwaggerProvider<oxfordApi>

let oxford = Oxford()
```

Connecting to the New York Times API is slightly more complicated. You need to create the same three lines of set up, but the New York Times requires the API key to be passed as part of the headers, so you have to use **CustomizeHttpRequest** to add it. After this, the two type providers act in the same way.

```
[<Literal>]
let topStoriesApi=
  "https://api.apis.guru/v2/specs/" +
```

Listing 3: Resulting JSON from calling the New York Times API

```
{
  "status": "OK",
  "copyright":
    "Copyright (c) 2017 The New York Times Company.
     All Rights Reserved.",
  "section": "travel",
  "last_updated": "2017-01-05T13:15:06-05:00",
  "num_results": 23,
  "results": [
    {
      "section": "Travel",
      "subsection": "",
      "title": "Hotels and Resorts to Travel to in 2017",
      "abstract": "Sometimes a hotel is just a place to sleep.
         Other times, it's a destination.",
      "url": "http://www.nytimes.com/2017/01/05/travel/
         hotels-and-resorts-to-travel-to-in-2017.html",
      "byline": "By ELAINE GLUSAC",
      "item_type": "Article",
      "updated_date": "2017-01-05T01:22:18-05:00",
      "created_date": "2017-01-05T01:22:19-05:00",
      "published_date": "2017-01-05T01:22:19-05:00",
      "material_type_facet": "",
      "kicker": "",
      "des_facet": [
        "Hotels and Travel Lodgings",
        "Eco-Tourism",
        "Spas",
        "Travel and Vacations",
        "Swimming Pools"
      ],
      "org_facet": [],
      "per_facet": [],
      "geo_facet": [],
      "multimedia": [
        {
          "url":
             "https://static01.nyt.com/images/2017/
              01/08/travel/08LODGING1/
              08LODGING1-thumbStandard.jpg",
          "format": "Standard Thumbnail",
          "height": 75,
          "width": 75,
          "type": "image",
          "subtype": "photo",
          "caption": "New hotel on Kokomo Island, Fiji.",
          "copyright": "Small Luxury Hotels of the World"
        },
      ],
      "short_url": "http://nyti.ms/2j6NFrS"
    }
...
```


```
topStories.Results
  |> Array.collect (fun r -> r.
<EOF>
```

Abstract	string
Byline	string
CreatedDate	string
DesFacet	string []
Equals	bool
GeoFacet	string []
GetHashCode	int
GetType	System.Type
ItemType	string
Kicker	string
MaterialTypeFacet	string
Multimedia	SwaggerTimesNYC.ArticleMultimediaItem []
OrgFacet	string []

```
val Abstract : string

From type:  String
Assembly:   FSharp.Core
```

Figure 4: Using IntelliSense to explore the New York Times Top Stories API

Listing 4: First pass of words contained in the abstracts of the top travel articles. The length of this array is 213 and, clearly, it needs some cleanup.

```
val results : string [] =
  [|"$12"; «100,000»; «125th»; «12th»; «150th»; «200th»;
    "2017"; «2017,»; "2017."; «20th»; «350th»; «A»; «Abu»;
    "Alliance"; «America»; «America's»; "Amsterdam";
    "Auditorium."; «Austen,»; «Australia.»; «Bahamas»;
    "Beijing"; "Big"; «Canada,»; «Canadian»; «Cape»;
    "Chicago"; «Club»; «Cup»; «Dhabi,»; "Dozens"; «Every»;
    "Five"; «For»; «From»; «Glimmers»; «Go»; «Harry»;
    "Here"; «Highlands,»; «Holland»; «Hoxton»; «In»;
    "Jane"; «Jonathan»; "Like"; «Line»; «London»; «Med,»;
    "More"; «Musical»; «New»; «Other»; "Places"; «Potter»;
    "Researchers"; «Ryman»; «Saskatoon,»; «Scottish»;
    "Sometimes"; «Swift»; «Tapawingo»; "The"; «This»;
    "Town"; «Travel»; «With»; "a"; «abound»; «activities»;
    "aid"; «airline»; «airlines»; "airplane"; "airport";
    "airways"; «all»; «allowing»; «along»; «alternatives»;
    "among"; "an"; «and»; «anniversary»; «annual»; «answers»;
    "anticipating"; «apps»; "apps."; «are»; «areas»; «art»;
    "as"; «aspects»; «attempt»; «attended»; "authors";
    "author's"; «avalanches»; «balancing»; ...|]
```

```
  "nytimes.com/top_stories/2.0.0/swagger.json"
type TimesNYC = SwaggerProvider<topStoriesApi>

let timesNYC = TimesNYC()

timesNYC.CustomizeHttpRequest <-
  fun req ->
    req.Headers.Add("api-key", keys.nytimes); req
```

Now that you've set up your connection to the APIs, it's time to get some data. Let's start by finding the top stories for the "Travel" section of the paper, in JSON format.

```
let topStories = timesNYC.Get("travel", «json», «»)
```

Using type providers, if you aren't sure in advance what the result type looks like, you can call the API and just explore.

That's it! You'll receive a response in JSON format similar to **Listing 3**, contained in a **TimesNYC.GetResponse** object. Now, check out **Figure 4**, and compare. The topStories value contains an array of Result objects. For each of the results, you have IntelliSense access to all of their properties: Abstract, Byline, CreatedDate, etc. Using type providers, if you weren't sure in advance what the result type looked like, you could call the API and just explore. There's no need to guess.

The goal, again, is to count the number of unique non-proper noun words used in the abstracts. You need to access the abstracts and then to split them up by words. This original array has 309 items in it, so make sure that the words are distinct. Let's also sort the list, for good measure.

```
topStories.Results
 |> Array.collect (fun r -> r.Abstract.Split(‹ ‹))
 |> Array.distinct
 |> Array.sort
```

This returns a rather messy set of data, with punctuation, dates, and proper nouns that aren't going to be in the dictionary, as you can see in **Listing 4**. You need to clean this up a bit before you can make use of the Oxford API. Let's define a simple **cleanup** method, using a regular expression to filter out any "words" that have numbers or a few special characters. If the word doesn't have any of the offending characters, there might still be some associated punctuation, so it's a good idea to remove that as well. The return type here is an Option type, which I've used in previous articles. This forces you to check whether the data is valid before being able to use it.

```
let cleanup (text:string) =
  if Regex.IsMatch(text,@"[0-9''$/–]") then
    None
  else
    Some(text.Replace(",","")
      .Replace(".","")
      .Replace("?","")
      .Replace(""","")
      .Replace(""",""))
```

You'll need to insert a call to your **cleanup** function, like so:

```
topStories.Results
 |> Array.collect (fun r -> r.Abstract.Split(‹ ‹))
  |> Array.choose (fun a -> cleanup a)
```

Listing 5: The findLemmas function

```
let findLemmas word =
  try
    oxford.GetInflections(
      "en",
      word,
      keys.oxfordId,
      keys.oxfordKey)
      .Results
    |> Array.collect (fun r -> r.LexicalEntries)
    |> Array.filter
      (fun l ->
        l.GrammaticalFeatures.[0].Text <> "Proper")
    |> Array.collect (fun l -> l.InflectionOf)
    |> Array.map (fun l -> Some(l.Id))
    |> Array.head
  with
  | ex -> None
```

```
|>Array.distinct
|>Array.sort
```

This helps to clean up your list of words quite a bit. You're down to 197 total items. But there are still some repetitions. For example, the list contains both "airline" and "airlines," which are really the same word. This is how the Oxford API can help you out. They provide a "Lemmatron" call, which will lemmatize your words for you. If you've not heard the term before, lemmatization replaces each word with its most basic form, without any extra prefixes or suffixes. So, both "airline" and "airlines" return the more basic form, "airline." You can still count them as one word!

Let's go through the code in **Listing 5** line by line to create the **findLemmas** function. Because you've already set up your connection to the Oxford API, you just need to make the call to the **GetInflections** method. This requires you to send along the language ("en" for English), the specific word you want to look up, and your credentials for calling the service. Rather than setting a value for the call and requesting the results in the next step as you did for the New York Times, it's easy enough to define the value as including the step into **Results** from the start. You're also starting a **try-with** block (similar to a **try-catch** block in C#) that you'll complete later.

```
let findLemmas word =
 try
  oxford.GetInflections(
   "en",
   word,
   keys.oxfordId,
   keys.oxfordKey)
   .Results
```

The Oxford API returns an array that's perfect to send through a pipeline. First, you want to map each result to the interior array of lexical entries. This returns an array because one spelling might mean several different things, even different types of speech. The API returns information for each one. By using **Array.collect**, the result type is an array, rather than an array of arrays, because it concatenates the resulting arrays.

```
  |>Array.collect (fun r ->r.LexicalEntries)
```

Next, let's filter the proper nouns.

```
  |>Array.filter
  (fun l ->
   l.GrammaticalFeatures.[0].Text <> "Proper")
```

Now it's time for the Oxford API to reveal whether the word is in its most basic form or not by calling **InflectionOf**. An inflection is a word that contains, for example, a different tense, person, or case, than the most basic form. This tells you what your word is an inflection of.

```
 |>Array.collect (fun l ->l.InflectionOf)
```

Next, **InflectionOf** returns both an ID and text, so you should take the ID (which is the lemmatized word) for each of the lexical entries that were returned. Again, you want to use an **Option** type to force a check for null data.

```
    |> Array.map (fun l ->Some(l.Id))
```

Finally, you return the first item in the array.

```
    |> Array.head
```

Now, it's time to complete your **try-with** block. In this case, it's easiest to drop every word where there's a problem, so you blanket catch all exceptions and return **None**.

```
 with
| ex -> None
```

Now, you just need to add a call to your **findLemmas** function, like so:

```
topStories.Results
 |>Array.collect (fun r ->r.Abstract.Split(' '))
  |> Array.choose (fun a ->cleanup a)
 |>Array.distinct
 |>Array.sort
  |> Array.map (fun w -> findLemmas w)
```

There's one problem in calling the Oxford API. They limit calls to 60 requests per minute. Because this code will request way over 60 items in a few milliseconds, the API responds by truncating the request to 60, and throwing errors for the rest. No good! You'll need to create one more function, one that splits your array into chunks of 60 words, waiting a minute between sending each group. Let's call it **splitToFindLemmas**.

```
let splitToFindLemmas (wordArray:string[]) =
 let size = 60
 [|for start in 0..size..wordArray.Length-1 do
  if start <> 0 then Thread.Sleep 60000
  let finish =
   if start+size-1 > wordArray.Length-1 then
    wordArray.Length-1
   else
```

```
    start+size-1
  yield
   wordArray.[start..finish]
   |> Array.choose (fun w -> findLemmas w)|]
 |> Array.concat
```

Let's look at this line by line again, starting with the declaration. You might have noticed that I needed to declare the type of the incoming parameter. F# has type inference, but every now and then it needs a little push. I found that in this set of code, I needed to specifically declare this type. There also tend to be two camps of F# programmers: Those who declare all types for clarity and those who omit as many types as possible for conciseness. Both are good options, I just happen to side with #TeamConcise. If you side with #TeamClarity, this is how you set up your declarations.

Because this code requests way more than 60 items in a few milliseconds, the API responds by truncating the request to 60, and throwing errors for the rest. No good!

```
let splitToFindLemmas (wordArray:string[]) =
```

Next, I declare a size value. It's not necessary, but it helps keep the code a little cleaner, because you refer back to it a few times.

```
 let size = 60
```

Most of the rest of the code is an array comprehension that is started with this next line. I'm declaring a value, **start**, which runs from 0 to the last item in the array, with steps determined by **size**, which is, in this case, 60.

```
 [| for start in 0..size..wordArray.Length-1 do
```

Next, a quick check. You need to sleep for a minute between each call, but there's no need to sleep for the first one. In F#, everything is an expression, and all paths of an **if** statement need to return the same type. In this case, you don't need an **else** statement because **Thread.Sleep** returns **unit**().

```
  if start <> 0 then Thread.Sleep 60000
```

Next up, I set up a **finish** value, which is the lesser of the array's upper bound or the next step.

```
  let finish =
       if start+size-1 > wordArray.Length-1 then
    wordArray.Length-1
   else
    start+size-1
```

Finishing off the array comprehension, I yield a slice of the array from the **start** value to the **finish** value and then call the **findLemmas** function for each of the words in that slice.

```
  yield
   wordArray.[start..finish]
    |> Array.choose (fun w -> findLemmas w)|]
```

Finally, I concatenate each of the slices, so that I only return one array.

```
 |> Array.concat
```

Now that that's all sorted, you return to your **topStories** pipeline and can add a quick call to **sortToFindLemmas** and another call to **Array.distinct** to remove the duplicates that lemmatization has created, and you're all finished! You have a final count of 104 unique words used in travel abstracts in the New York Times' top stories.

```
topStories.Results
 |> Array.collect (fun r -> r.Abstract.Split(' '))
   |> Array.choose (fun a -> cleanup a)
 |> Array.distinct
 |> Array.sort
   |> sortToFindLemmas
 |> Array.distinct
```

Where to Learn More

Many of the type providers that I shared with you today are available as part of the **FSharp.Data** project: XML, HTML, and JSON. You can find more information and documentation about them here: https://github.com/fsharp/FSharp.Data. The Swagger type provider has also been open sourced, and more information is available here: http://fsprojects.github.io/SwaggerProvider/.

Rachel Reese

CODE

How to Know Your Team is Productive and Delivering Quality

Every software team that takes pride in its work likes to believe that it's delivering high quality with high productivity. It's easy to say that a team is highly productive, and anyone can claim that the software he produces is of high quality. But our industry hasn't adopted clear metrics for either quality or productivity.

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

Jeffrey Palermo is the CEO of Clear Measure, an Austin-based software engineering firm that serves as fractional software departments for companies in non-technology industries. Jeffrey has been recognized as a Microsoft MVP for 11 consecutive years and has spoken at national conferences such as Tech Ed, VS Live, and DevTeach. He's an author of several books and articles, and loves to share knowledge with business leaders. A graduate of Texas A&M University, an Eagle Scout, and an Iraq war veteran, Jeffrey likes to spend time with his family of five out on the motorcycle trails.

The industry certainly has adequate research to define and report these metrics, but most software leaders either don't know about the research or haven't applied it. This article covers:

- Estimating a team's current throughput
- Tracking defect removals
- Crafting a high-quality development process
- Measuring productivity and quality

Estimating Current Productivity and Quality

With any of the major software project tracking tools, you can analyze the work as it moves through the system and gain a good understanding of the productivity and quality the team is delivering. For example, Visual Studio Team Services (my instance is at https://clearmeasure.visualstudio.com/) supports a process where work moves through various stages until the work is marked as complete.

This article uses a sample software department with anonymized data to illustrate a technique for measurement. By analyzing the historical work tracked across multiple projects, we've been able to produce results that can be very useful to software leaders charged with managing teams delivering software. As we examined the available data stored within VSTS, we conducted a quantitative analysis that can be used to estimate the productivity and quality that a software team produces.

Throughput Detection

To detect information about current throughput, we examined the movement of work through various existing stages in VSTS. We observed that although each project used differing stage (or work item status) names, we were able to map each project to a standardized set of stages and perform some conclusions that were meaningful. For example, we were able to harvest some high-level information about throughput in the software development process. **Figure 1** shows a graph by project that reveals the relationship of the time spent progressing the work forward to the time spent waiting on others.

In this chart, we were able to determine the scalar average and relationship between the average lead time and the actual work time within that lead time. We found that the actual time working on an item is less than the total time the item was in the overall process. This chart breaks up these data by project. We can see the relative pace of work between projects but also that Proj6 has the greatest amount of time work is waiting for someone to work on it. When we measured this for the first time, we couldn't draw any conclusions immediately. Rather, we used it to verify that we were able to begin drawing correlations and meaningful metrics from the existing work-tracking data. This gave us confidence that when the department implements process changes, improved results can be tracked and reported easily in a similar fashion, and we will see a reduction of wait times. In addition, we inferred some process differences between projects. Consider the chart in **Figure 2**.

In the chart in **Figure 2**, we can see that some projects incurred a greater amount of work that is almost completed but waiting for release into production. For example, Proj7, Proj2, and Proj1 all had noticeably more work waiting on deployment than the other projects. In fact, Proj3 appears to deploy to production as soon as something is ready. But Proj3 also has the greatest lead time in the "backlog" stage. In other words, there's a greater percentage of items waiting to be worked at all, but once they're begun, they finish and deploy quickly. As a software leader begins to measure this type of information for the first time, he sees big differences in team and project behavior. As time progresses and various projects and teams execute with a more consistent workflow, the software leader sees the flow of work begin to look more similar.

Defect Detection

In addition to identifying information about throughput of work through the software development process, we were also able to deduce a rudimentary quantification of defects that were found during normal development. According to industry research (explained below), we know that internal software teams need to find and remove 85% of the defects that will naturally occur during software development and prevent them from being released to users in a production environment. Some defects are explicitly logged in a tracking system and are visible through reports, but in an agile process, there are also a statistically significant number of defects that are caught and corrected quickly without ever being entered as a formal defect work item in VSTS. If these aren't included, you might erroneously conclude that defect removal efficiency is below 85% because of the underestimation of defects that are removed before production. The chart in **Figure 3** illustrates this finding.

When we examine the history of each work item, we can see when the status moves backward rather than the normal movement forward toward deployment to production and being marked "done." When the status of a work item is reset to a previous status, we can infer that there's a problem. Even though a defect might not be entered in the system, we count this as an implicit defect. This technique

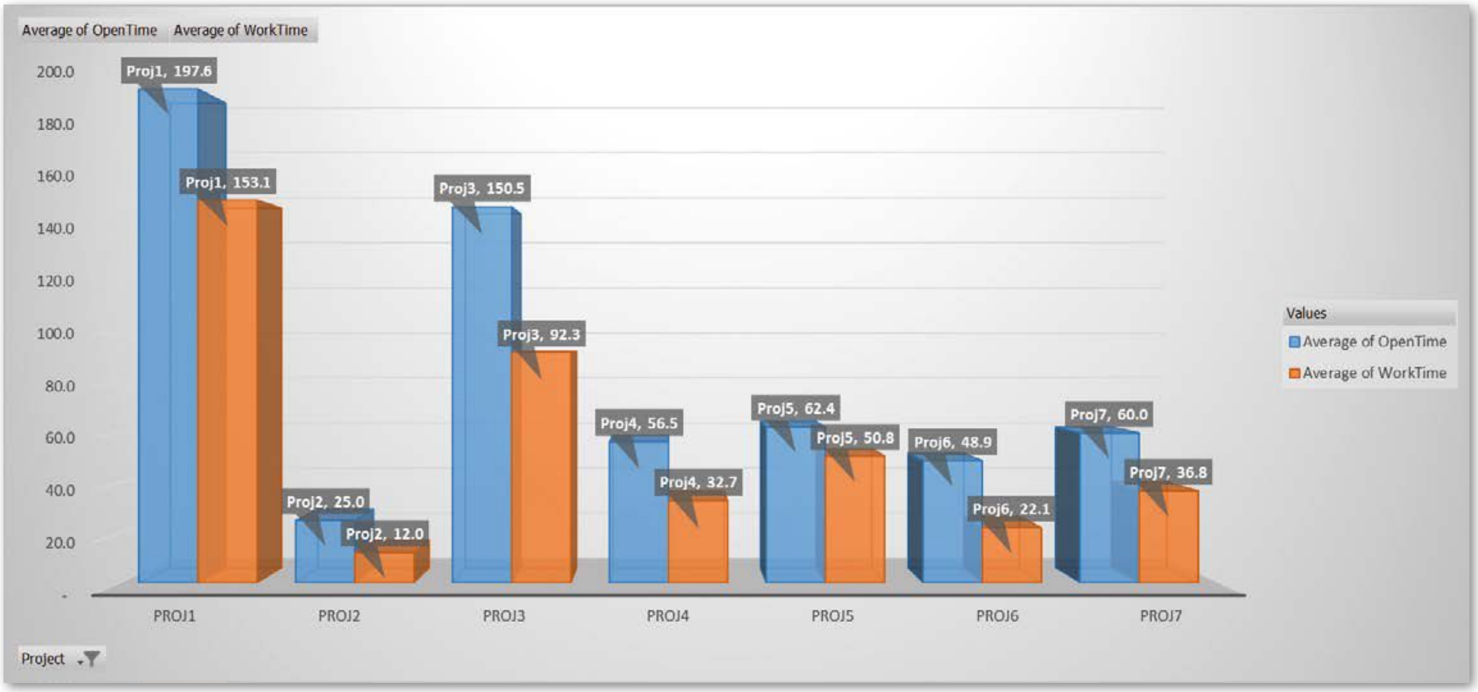


Figure 1: Open time versus work time by project

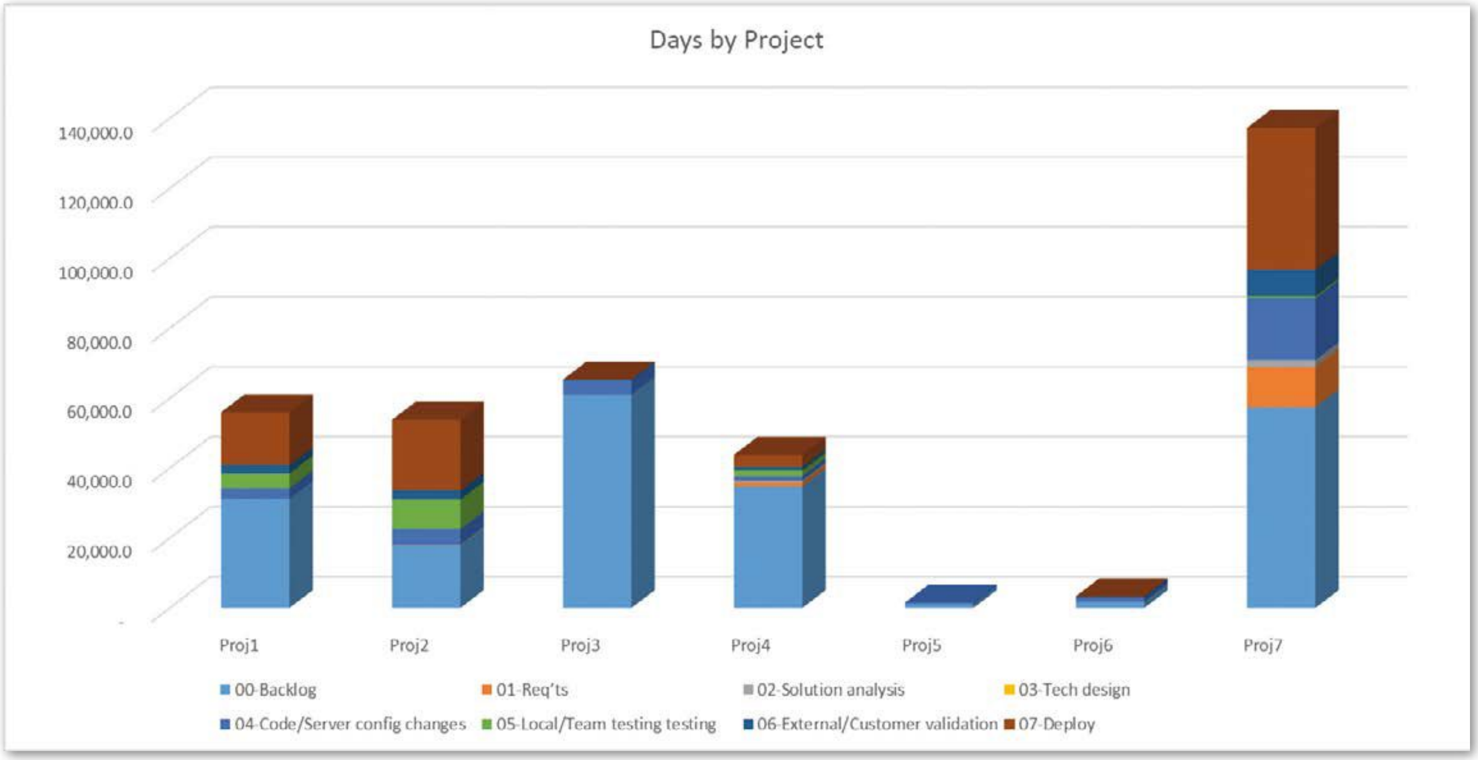


Figure 2: Time by stage in each project

can accommodate agile teams that move work through to production very quickly and very often. As we can see, we have a significant number of items (stories) that move to External/Customer validation and then get bumped back to Local/Team testing because of some needed rework. It's a good outcome to catch defects in any manner possible rather than delivering them to the customer through production. And using this method, among others, we're able to quantify caught and resolved defects that would otherwise go unreported in metrics on a scorecard.

Constructing a Software Development Process

There are as many implementations of the software development lifecycle (SDLC) as there are companies that create custom software. Some companies have additional processes as necessary, but it's hard to get by without the following essential four:

- **Plan:** Decide what will satisfy the market and prepare to dispatch desired capabilities to personnel who can build the requested functionality.
- **Build:** Put ideas into released software that works properly.
- **Operate:** Render the services of the software to the users or customers and support them. Ensure stability of the functionality released to production.
- **Maintain:** Execute service schedules, break/fix, and repairs to ensure that release functionality is repaired quickly and that preventative measures are taken to prevent future breakdowns.

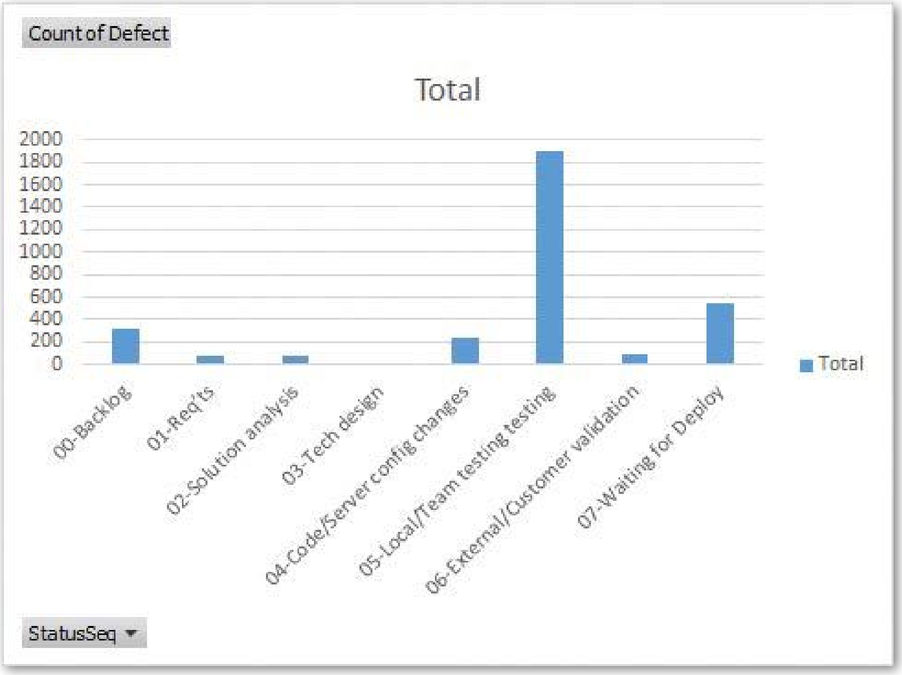


Figure 3: Defects by stage can be used to estimate quality

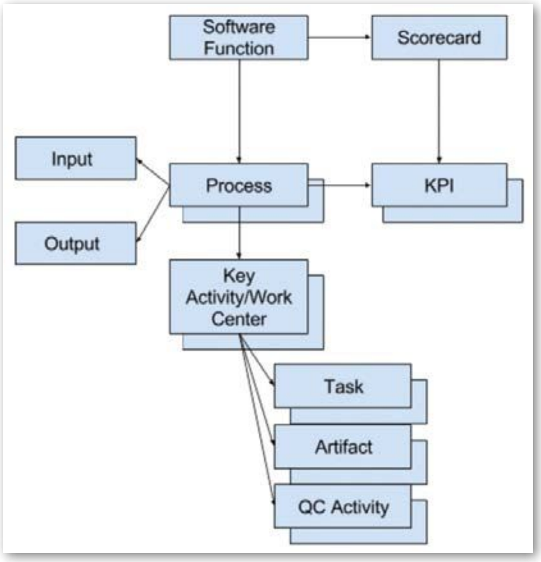


Figure 4: The shape of a measurable software process

With these being the essential four processes of a software development function, each follows a very similar model while being very different processes.

The Shape of a Process

Once you understand the shape of a process within a software development function, you can begin to design metrics, Key Performance Indicators (KPIs), and a scorecard that can be used in a consistent way across software projects and teams. The model in **Figure 4** shows the shape of a process within the software development function.

The software function has a single scorecard. The scorecard is made up of multiple KPIs, each owned by a process within the software function. Each process has an input and an output. The output is created, added to, and refined by each key activity (or work center) in the process. Each key process, in turn, has multiple tasks identified in order to produce an artifact that's added to the output of the process. Each key activity is arranged in linear order with the artifact of one serving as an input for the next. Tasks create and refine the artifact, and quality control activities identify defects to be removed before the artifact is accepted in a downstream key activity.

Principles the Process Must Implement

"The Phoenix Project," by Gene Kim is recommended reading for anyone seeking to implement a high-performance software delivery process. In his book, Kim explains three key principles to put into play when implementing processes in a software and IT environment. Kim calls them the "three ways."

The first principle is about the type and flow of work that flows through the process. The work must take on the following characteristics and behavior:

- Single-piece flow is ideal, but small batches are required.
- Work batches should be structured so that they can be completed as quickly as possible, leading to small work intervals.
- Defects should be caught and corrected at the source and never passed downstream in the process.
- The KPIs resulting from the work should be process KPIs and not resulting from metrics from a single key activity or work center. In other words, speed of coding is not a valid metric because it's only a piece of a process.

The second principle is about feedback loops from activities further down the process so that future defects can be prevented. Examples of this are:

- Immediately sending a piece of work back to the work center responsible for the defect.
- Creating automated quality control activities to immediately perform automated testing, inspections, and static analysis in order to catch defects quickly.

The third principle aims to create a culture of continuous improvement and mastery. For continuous improvement of the process, this principle encourages us to:

- Create improvement cycles where the team must make an improvement, even if very small.
- Take risks and experiment in order to find that next additional improvement at every cycle.
- Use time when things go wrong to flow feedback into our cycle of improvement.

Additionally, in Jez Humble's book, "Continuous Delivery," Humble outlines eight principles and eight practices for teams that continuously deliver valuable software to customers. The eight principles are:

1. The release process must be repeatable and reliable.
2. Automate everything.
3. If something is difficult, do it more.
4. Keep everything in source control.
5. "Done" means released.
6. Build quality in.
7. Everyone is responsible for the release process.
8. Improve continuously.

The four practices are:

- Build binaries only once.
- Deploy to each environment the same way.
- Smoke test your deployment.
- If anything fails, stop the line.

Measuring the Development Process

Once a linear one-way process is assembled and executing, two aspects of the process to continually measure and optimize are:

- Quality
- Throughput

Each should be measured for actuals and forecasted for the future. In addition, these metrics can be turned into a KPI by adding a threshold, or acceptable levels:

- Poor
- Okay
- Good

The thresholds can change periodically to align with organizational goals. Industry benchmarks and available research can be used to inform the business decision about where to place thresholds.

Capers Jones has probably the most comprehensive book on software measurement and metrics available in the industry. In his book, "Applied Software Measurement," he analyzes and reports on scores of software studies across languages, platforms, project types, and methodologies. In addition, in another of his books "Software Engineering Best Practices," Jones reports on the 50 best software development practices and the 50 worst software development practices based solely on results from research, thereby producing a quantified listing of practices.

Quality Control

In "Software Engineering Best Practices," Jones reports data indicating that a high-quality process can be low-throughput but that a low-quality process cannot be high-throughput. Therefore, high quality must come before high throughput, and high throughput is not achievable without high quality, most probably because as a team generates a high level of defects, much time is consumed just fixing problems that the process generates.

In addition, research indicates that three quality control activities have the best track record of helping an organization achieve 85% Defect Removal Efficiency (DRE) when combined. There are:

- Inspections
- Testing
- Static analysis

Anything done manually is more error prone and slower than its automated counterpart; therefore, you desire to implement the automated form of each quality control method if possible.

In addition to discovering and removing defects, the data from the research of Capers Jones, as well as the International Software Benchmarking Standards Group (ISBSG), indicate that teams that take measures to prevent defects end up with better overall quality and higher throughput.

The modern concept described is technical debt. Technical debt is essentially unfinished work. It can be work that was deferred to future releases or it can be defects that are marked as deferred. Technical debt can also be unmitigated risks that stay around as organizational liability. Regardless of the source, increasing the quality of a process requires technical debt to be dealt with using the following activities in priority order:

- Identify and record the debt.
- Modify the process so that technical debt is prevented in the future.
- Pay the debt down during each improvement cycle.

The final quality concern that a high-throughput process must include is how to treat non-code defects. In "Software Engineering Best Practices," using data from Software Productivity Research and Namcook, Jones shares research that demonstrates that more defects occur before coding begins than in coding itself.

Two key metrics are critical to the measuring of quality in a software development process:

- **Defect potential:** The probable number of defects that will be found during the development of software applications
- **Defect Removal Efficiency (DRE):** Refers to the percentage of defect potentials that will be removed before the software application is delivered to its users or customers.

The above is explained in great detail by Capers Jones in his article, *Measuring Defect Potentials and Defect Removal Efficiency*. In fact, the research has yielded some interesting averages for both. **Table 1** illustrates what you should expect on a typical business application in

Defect Origin	Defect Potential	Removal Efficiency	Defects Remaining
Requirements defects	1.00	77%	0.23
Design defects	1.25	85%	0.19
Coding defects	1.75	95%	0.09
Documentation defects	0.60	80%	0.12
Bad fixes	0.40	70%	0.12
Total	5.00	85%	0.75

Table 1: Defect research application to the United States

the United States. With the average for defect potentials being about five defects per function point, you should expect about five defects for every 52-54 lines of a high-level computer programming language.

Additional research data also reveals that formal inspections can average 65% defect removal efficiency, and testing can only identify 30-35% of defects, but when combining formal inspections with testing and automated static analysis, organizations can reliably achieve 85% defect removal efficiency.

Productivity and Throughput

Productivity, from a metrics standpoint, is a measure of how many units of work an organization pushes through its process. It can also be called throughput. Measuring throughput for a software project can be very similar to measuring throughput of a factory or manufacturing plan, and many of the same tenets of lean manufacturing can apply. In a software development process, there's one economic measure of throughput that's supported by several decades of research:

- Function points per person-month
  - Useful on a per project basis
  - Insufficient for closely monitoring throughput during development

Because the industry's body of productivity data is measured in Function Points or variations of function points, that's the unit available for comparing productivity across the industry. For an organization that doesn't count function points before building software, function points can be estimated by backing into it using lines of code after the software is built. Although leaving room for some inaccuracy, this practice provides a reasonably close approximation useful enough for comparison across the industry's body of research. For example, knowing that 52-54 lines of a high-level programming language are typically required to implement one function point of functionality, you can back into a function point estimate that can be used to calculate productivity per person-month.

Although this measure allows you to create an economic sampling perhaps once per quarter, you need some measures that can be sampled more frequently, even weekly. For that purpose, use the following:

- Number of features exiting the process
- Number of features in the process per team member
- Number of business days on average that a feature is in the process as well as in a stage

With these metrics, you can always know:

- Total throughput
- Work in progress at any time
- Speed of work through the process and bottlenecks

The goal is an optimal number of features exiting the process according to the ability of downstream processes to accept the work. And with the two supporting metrics, you can determine optimal staffing and lead times for work making it through the process.

Conclusion

Every software department wants to deliver quickly and deliver a quality result. With the research available in the industry, as well as the measurement techniques demonstrated in this article, software leaders can begin to measure projects. If a measurement has never taken place, the first numbers will be disappointing. But with continual process improvement, any team can achieve the removal of 85% of potential defects before production release and create a process that smoothly moves software feature requests through multiple distinct stages and on to production without bottlenecks.

Jeffrey Palermo

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

Exploring Sticker Packs with iOS 10

Jason Bender
Jason.bender@rocksaucestudios.com
www.jasonbender.me
www.twitter.com/TheCodeBender

Jason Bender is the Director of Development for Rocksauce Studios, an Austin, Texas-based Mobile Design and Development Company. He's been coding since 2005 in multiple languages including Objective-C, Java, PHP, HTML, CSS, and JavaScript. With a primary focus on iOS, Jason has developed dozens of applications including a #1 ranking reference app in the US. Jason was also called upon by Rainn Wilson to build an iOS application for his wildly popular book and YouTube channel: SoulPancake.

Communication is an essential part of our everyday lives and, as technology has progressed, the way we communicate has also evolved. With the advent and rise in popularity of emojis, for instance, communication has begun to expand beyond verbal and written forms Into something more visual. In fact, 92% of the online population uses some form of emoji (Emogi marketing report 2015, http://www.adweek.com/socialtimes/report-92-of-online-consumers-use-emoji-infographic/627521).

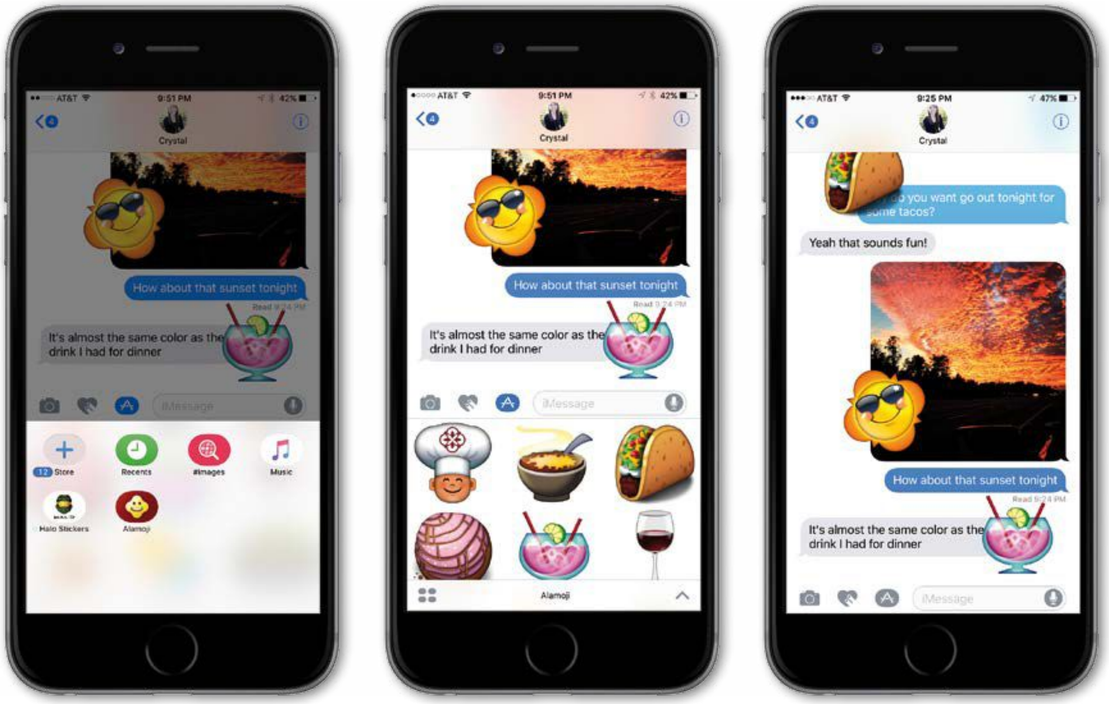
With the release of iOS 10, Apple has created a new medium for visual communication using something they call stickers. Stickers work similarly to emojis but carry with them additional functionality that result in a more interactive experience. In this article, you'll take a closer look at stickers, sticker packs, and the newly accessible iMessages App Store.

What are iOS Stickers?

With the introduction of stickers in iOS 10, users now have a new way to express themselves within their existing iMessage conversations. Stickers are static or animated images that can be tapped and sent in a conversation thread just like you would a normal emoji. However, stickers have additional functionality baked in. For instance, you can peel a sticker, resize or rotate it, and place it on top of or next to message bubbles in the conversation flow, within photos, or even other stickers. See **Figure 1** for a sample conversation using stickers.

Sticker Packs

1. Alongside the announcement of stickers, Apple also introduced the dedicated iMessages App Store; your one-stop-shop for applications that run as extensions within the default iMessages application. Currently, two types of iMessage apps exist: Sticker Pack and custom.
2. A Sticker Pack application is a limited-functionality iMessage app that delivers a single grouping of stickers. A custom iMessage application removes the limits of the basic Sticker Pack template, giving you the ability to layer on additional functionality as you see fit (I'll cover this approach in more detail later in the article).
3. **Figure 1** demonstrates what a Sticker Pack application looks like running on the device. Notice that the far-left screen depicts the listing of iMessage applications currently installed on your device. The middle screen displays what it looks like if you tap on an installed Sticker Pack application, and the far-right screen demonstrates a conversation using the stickers from the pack.
4. In order to use the stickers, users interact in one of two ways. If you tap on a sticker, it adds the sticker to the message entry field so that you can send it just as you would a normal emoji. However, if you tap and hold on a sticker, it peels off of the keyboard and allows you to manipulate it and place it at your desired location within the conversation window. Users within the conversation can tap and hold on a sticker to see details about it, including which iMessage app it originated from.



How about that sunset tonight
Read 9:24 PM
It's almost the same color as the drink I had for dinner
o you want go out tonight for me tacos?
Yeah that sounds fun!

Figure 1: The Alamoji iMessage Sticker Pack, available for download in the iMessages AppStore, is used here to demonstrate sticker pack functionality.

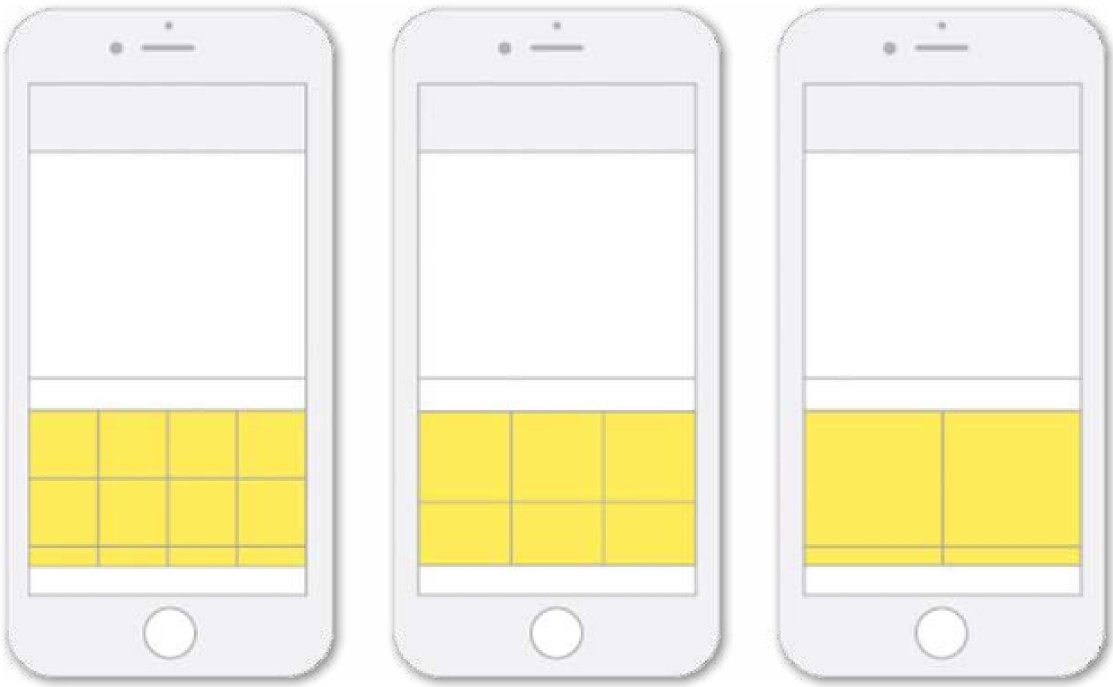


Figure 2: Sticker placement examples based on the size chosen, with small on the left, regular in the middle, and large on the right.

Sticker Pack Icon Guidelines

For additional details on icon requirements for Sticker Pack Applications, please visit: https://developer.apple.com/ios/human-interface-guidelines/extensions/messaging/

Setting up Xcode

For help setting up Xcode, visit: https://www.appcoda.com/learnswift/get-started.html

So Easy a Caveman Could Do It

Because Sticker Pack applications are a standard iMessage application type, XCode has a specialized template for building them. This template contains everything you need to construct and deploy a sticker pack to the iMessages App Store. In fact, to build one, you don't have to write a single line of code. The template contains all of the functionality, only needing you to add in your stickers and the application icons.

With the introduction of stickers in iOS 10, users can now peel a sticker, resize it, and place it on top of or next to iMessage conversation bubbles, photos, or even other stickers.

The only thing you really need in order to build a Sticker Pack application is the appropriately sized sticker assets. It's important to note that you're limited in the sizing options you have to choose from. You can choose one of three sizes, as defined next and also depicted in **Figure 2**.

- Small: 300px by 300px
- Regular: 408px x
- Large: 618px by 618px

In **Figure 2**, the right image depicts a small sticker size and how the Sticker Pack template formats them four to a row. The middle image demonstrates the formatting of a regular size sticker at three to a row, similar to the

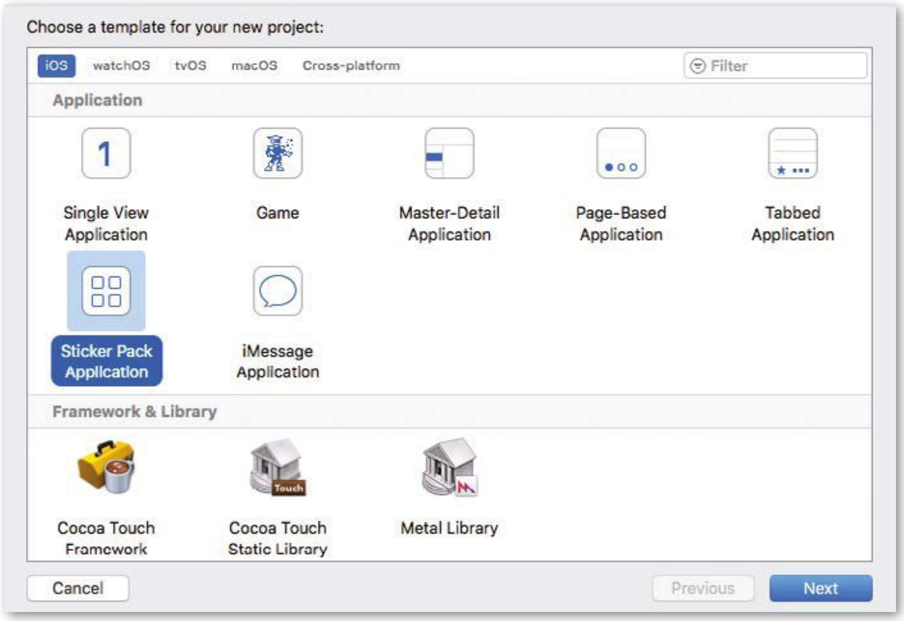


Figure 3: Choose the Sticker Pack Application template in Xcode

Sticker Pack application demonstrated in **Figure 1**. The right image shows a large size sticker at two to a row.

Apple also released a set of guidelines for creating the assets you use as stickers. Each sticker should adhere to these regulations:

- For static stickers, you must use of one of the following formats: PNG, APNG, GIF, JPG
- For animated stickers, you must use one of these formats: APNG or GIF
- A single sticker must not exceed 500KB

It's worth noting that Apple specifically recommends PNG and APNG formats and warns that PNGs saved in XCode uses a 24-bit palette by default, which could translate into larger-than-expected file sizes.

You also need to provide a set of various-sized application icons that will represent your application across the OS. As you might have noticed from the left screen in **Figure 1**, iMessage application icons are almost oval shaped. As a result, they have different size requirements and guidelines than a standard iOS application icon. You can find information relative to those sizes and the corresponding icon guidelines using the links provided in the sidebar of this article.

Making Your Own Sticker Pack

Now that you have a basic understanding of iOS 10 stickers, let's take a look at the process of setting up and populating your own sticker pack application using Xcode 8.2.1 on iOS 10.2. For the purposes of this demonstra-

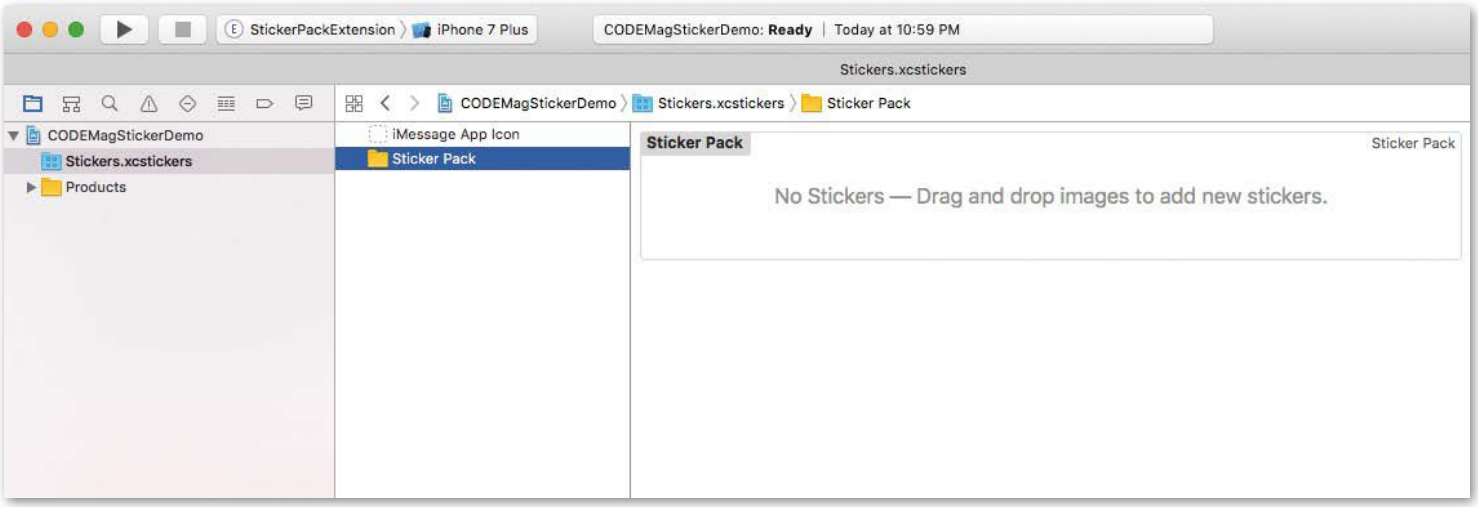


Figure 4: Empty Sticker Pack Application template

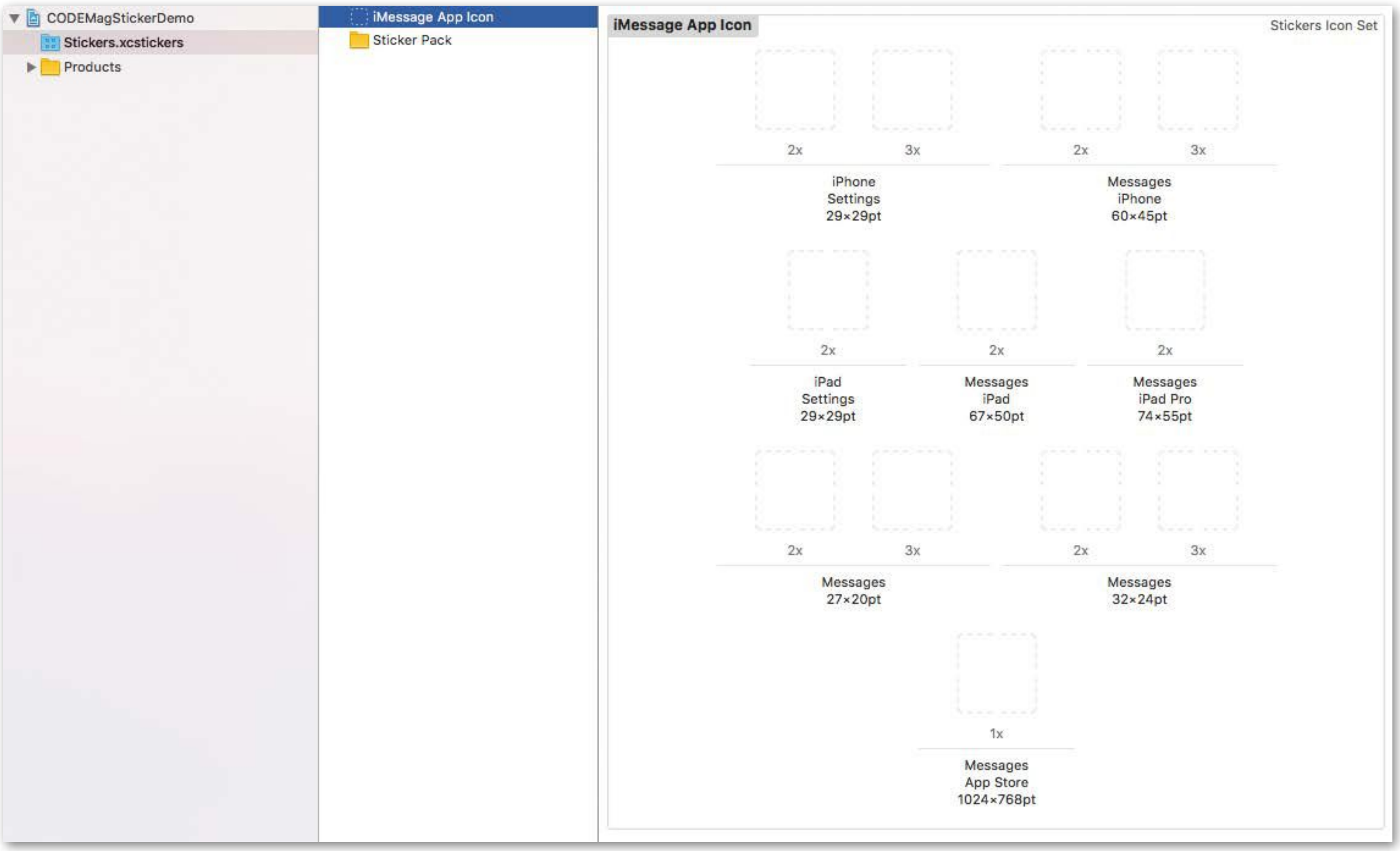


Figure 5: Add the proper application icon sizes to the Sticker Pack app.

tion, I'll assume that you have Xcode installed. If you need assistance installing or setting up the development IDE, check the sidebar for additional details.

Getting Started

To start, open up Xcode and select "Create a new Xcode Project" from the Welcome modal. Once you make this selection, you see a new modal resembling the one shown in **Figure 3**. This modal lets you choose the starting point for the application type you want to build. Make sure that iOS is selected at the top of the modal, select **Sticker Pack Application**, and then click Next.

Once you've made your template selection, Xcode prompts you to enter an application name and choose a team for provisioning. Provisioning the application for submission to the App Store is by far the most difficult part of the process and requires a working knowledge of the same provisioning and certificate procedures used to submit full iOS applications. For more detailed instructions on provisioning your Sticker Pack, refer to the sidebar. Once you've filled in the appropriate details, click Next and choose a location to save the project. When you're finished, Xcode launches the template you just created. You'll notice that the project is bare. It contains a target and a single assets folder named **Stickers.xcstickers**, as shown in **Figure 4**.

Drop the stickers you want to power the application template into this assets folder. Once you've composed your images, made sure that the sizes are uniform and correct, and verified that the file types match the requirements, you can drag the lot of them into this folder. Once you've done that, Xcode arranges them in a grid format within the asset folder's Sticker Pack subfolder. You can drag the icons around to re-arrange the order in which they appear in the application.

You can tell the template what size of sticker to display using the attributes inspector on the right-hand panel within Xcode. Xcode can detect the size manually but it also has rescaling functionality built in by default. That means that if you upload a large-size icon but want to display it as a small sticker size, you can do that. However, it's recommended to provide the exact size that you wish to display, as scaling is an unnecessary hit on performance.

Once you add the application icons, set the size of your stickers, and arrange the sticker order to your liking, all you have to do is build it, test it, and deploy it.

The last step is to add the application icons to the template. Within the **Stickers.xcstickers** asset folder, there's an **iMessage App Icon** section. If you select that, you'll see a list of sizes, as shown in **Figure 5.** You need to create the icons to the specifications detailed in the sidebar of this article, and then drag each one to its appropriate location in this panel. Once you add the application icons, set the size of your stickers, and arrange the sticker order to your liking, all you have to do is build it, test it, and deploy it.

Extending the Functionality

Although Sticker Pack Applications are simple to create, their functionality is rather limited. For instance, what if you want to have multiple categories of stickers, in-app purchases, or maybe the ability to edit a custom sticker? If you want to extend the capabilities of the default sticker application, you'll have to create a custom application using the default **iMessage Application** template in Xcode. This template gives you a special MSMessagesAppViewController object to serve as the entry point of the application, but that controller is a blank slate. You don't get any functionality baked in from the start because iMessage applications can be built to do any number of things and aren't limited to stickers. Therefore, building additional features requires the corresponding coding knowledge and background to do so.

Currently, you can build an iMessage App in either Objective-C or Swift. You can download Apple's official iMessage application sample at https://developer.apple.com/library/content/samplecode/IceCreamBuilder/Introduction/Intro.html.

Wrapping Up

Over the course of this brief article, you gained a basic understanding of what iOS 10 stickers are and how easily you can create your own sticker packs. If you're interested in a more complex look at stickers or iMessage applications in general, be sure to refer to the sidebars for this article for additional resources.

Jason Bender

Provisioning Your Application

For more detailed information about how to generate the needed profiles to provision your application, visit: https://developer.apple.com/library/content/documentation/IDEs/Conceptual/AppDistributionGuide/MaintainingProfiles/MaintainingProfiles.html

Advanced iMessages App Tutorial

To learn about the endless possibilities of custom iMessage apps, check out this tutorial for creating an interactive tic-tac-toe game in iMessage http://www.techotopia.com/index.php/An_iOS_10_Interactive_Message_App_Tutorial

Azure Skyline: Building and Deploying Services to Azure with .NET Core and Docker

As a professional .NET developer, writing apps for Linux probably hasn't been on your radar. Why would you even want to? Linux is for hipsters with MacBooks who write in Java and for some inexplicable reason, prefer to work in arcane and garishly colored command shells, right? All kidding aside, you're probably asking where this all fits in to the corporate world in which so many of us make our livings. Not including the large number of potential customers who prefer a non-Windows infrastructure, we'll be seeing more and more of Linux in the Microsoft-centric world and the reasons are mainly economic. Customers with large infrastructures need fault tolerant, scalable, secure, and inexpensive environments. In short, they need proactive, robust DevOps without breaking the bank.

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.

Why Linux? Hasn't it been proven that the Total Cost of Ownership (TCO) of Windows is less than Linux? About four years ago, an open source project called Docker launched and changed the IT world. The concept wasn't new, but the implementation was good and began to take hold. Docker allowed us to create runtime sandboxes called containers to deploy our apps in. Once configured for the container, the apps ran happily without knowing or caring about anything outside of the container. This allowed the containers to be run on any computer. They could be moved around, scaled out and made fault-tolerant by creating multiple instances behind load balancers and treating them like a commodity. Apps could be bundled together in the same container or split up into separate containers. Imagine a system with 50 micro-services, each one being developed and deployed independently. You could run all 50 on a single computer or run one instance each on 50 computers or 50 instances on one computer and another 50 on a second computer, or x instances on y computers. Now imagine being able to get any of these scenarios up and running in a couple of hours and being able to change from one scenario to the next in even less time. That's pretty powerful stuff and when you consider that each of these containers only requires a few MB and can run in tiny amounts of memory compared to a big, fat .NET application, you can see why everyone's been talking about Docker.

Until a few months ago, Docker and Linux didn't generate a lot of interest for .NET developers. If you couldn't write apps for Linux, why would you care? All of that changed when three things happened. First, Docker worked with Microsoft and started allowing us to host containers on Windows 10 and Windows Server 2016. In fact, they introduced two types of container hosting, one similar to Linux and one that Linux users didn't have. It's based on Hyper-V and, although much heavier in terms of resource requirements, provides much better isolation between containers and thus, much better security. Second, Docker started supporting Windows Server Core and Nano containers (on Windows Docker Host computers), so our apps could run in Windows containers on Windows hosts. Yes, that's right! We can run IIS and the full ASP.NET stack as well as other Windows software in containers now. Although Windows containers tend to be a lot bigger and heavier than their Linux counterparts, we can now begin to do a lot of the same DevOps tricks that our Linux counterparts have been enjoying. We can for example, package our websites in containers running Windows and IIS and ASP.NET in a container and do the same for our Web API micro-services and enjoy the benefits of Docker.

Then, there's that third thing that happened. Microsoft created something called .NET Core that runs not only on Windows, but on Macs and Linux and a host of other platforms including Internet of Things (IOT) devices, tablets, and phones. Given the choice between writing C# programs that only run on Windows computers or writing the same program that can run on Windows, Linux, Mac, or a Raspberry Pi, why would I choose to only run on Windows? ASP.NET Core is new and just hit v1.1. It's not as complete or robust as the full .NET stack. You can't program desktop UIs in it (at least not yet). Although you can build some basic websites with .NET Core, it's not nearly as powerful as its big brother, but the services portion is looking good and that's one place .NET Core shines. Services in .NET Core are almost identical to services in ASP.NET Web API. In fact, I've copied and pasted quite a lot of my existing Web API code into .NET Core services and had it work perfectly on the first try.

If I can run my services on the smaller, faster, cheaper Linux containers and not only make my way into that large, untapped market but also offer those benefits to my existing clients at no extra cost to me, why wouldn't I embrace it? I can always run the very same code in Windows containers or on Windows proper if my client prefers it.

Create a .NET Core App

How can you get all this goodness too? First, you have to have either Windows 10 Professional or Enterprise, or Windows Server 2016 Essentials, Standard, or Data Center. And you have to be running a 64-bit OS. Go to https://www.docker.com/products/docker#/windows and download the appropriate Docker for Windows for your OS. The installation is simple and you can easily ac-

Create/Design/Develop/Connect AnDevCon

AnDevCon is the technical conference for professional software developers and engineers developing for the Android platform.

From learning the basics in an Android development bootcamp to embedded Android, Artificial Intelligence, and cross-platform technologies, AnDevCon offers the most comprehensive program with countless sessions and networking opportunities.

Great classes, the reception is awesome and there's better technical content than Google I/O!
— Kevin Cousineau, Mobility Architect, Ideas Improved

3 Big Tracks!

Android Essentials
A deep dive into the fundamentals all the way to the most advanced capabilities of Android and the Android ecosystem.

Cross-Platform Development
This track goes beyond native Android development to teach developers to master cross-platform development tricks, techniques, and tools.

Machine Learning/AI
An entire track at AnDevCon is devoted to machine learning and AI practices that are at the forefront of the next wave of Android and mobile technology!

- Hands-on Android development training
- Bring your unique problem, find your solution
- Learn from Google experts
- Be a part of the community

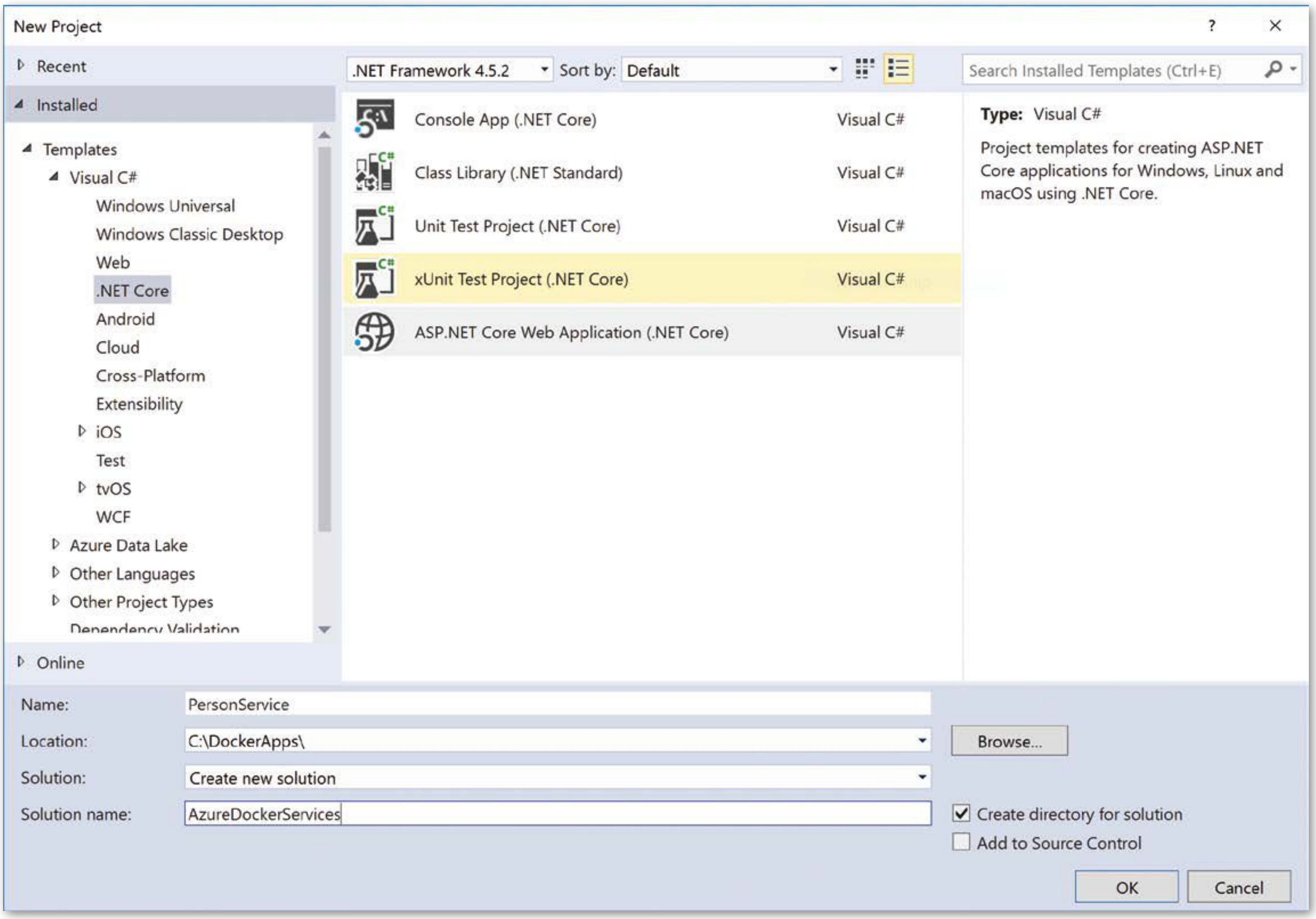
AnDevCon
The Android Developer Conference

July 17-19, 2017
Washington, DC

Register Early and SAVE!

www.AnDevCon.com

A BZ Media Event



.NET Framework 4.5.2 · Sort by: Default

Console App (.NET Core)	Visual C#
Class Library (.NET Standard)	Visual C#
Unit Test Project (.NET Core)	Visual C#
xUnit Test Project (.NET Core)	Visual C#
ASP.NET Core Web Application (.NET Core)	Visual C#

Type: Visual C#

Project templates for creating ASP.NET Core applications for Windows, Linux and macOS using .NET Core.

Name:	PersonService
Location:	C:\DockerApps\
Solution:	Create new solution
Solution name:	AzureDockerServices

☑ Create directory for solution

☐ Add to Source Control

Figure 1: Create a .NET Core REST services project

cept all the defaults. Next, you're going to want Visual Studio 2017. As I write this, it's in Release Candidate (RC) status. When you install, make sure to check the **NET Core and Docker** component. Of course, you don't need Visual Studio 2017 to build .NET Core apps; you can do that with Notepad and the .NET Core SDK, but we're not savages! We're used to good tools. This is all you need to build a service and run it in a Linux container.

We're not savages!
We're used to good tools.

Under Visual Studio's File menu, choose New Project. Under the Visual C# Templates, Choose ASP.NET Core Web Application (.NET Core). (You can see this in **Figure 1**.) Notice that Visual Studio doesn't allow you to set the framework version here (yet). It's very important in this version that you create the solution on your OS drive (normally C:\). Otherwise, you may run into a bug after installing Entity Framework Core NuGet packages where the project file no longer loads. If you install on a different drive and run into this bug, try moving the project onto your OS drive.

Next, choose the Web API template, set it to No Authentication, and check the Enable Container (Docker) Support checkbox, as shown in **Figure 2**.

Open the Output window and pin it open. Also, make sure you have an Internet connection. Notice that the Run button says **Docker**. Press it to begin the build and run process. The build is going to be a bit unusual and you'll notice that it can take a long time the very first time you build. By choosing to Enable Container (Docker) support, you asked Visual Studio to create a container image for you and to run and debug the code inside the container. You'll see the Docker command lines being run in the Output window. If the build fails, check your Docker Settings and make sure that the drive you're using for your source code is shared.

After compiling the code and setting up an internal network for the computer and the containers to communicate, you'll see a line that looks something like this: **Step 1: FROM microsoft/aspnetcore:1.0.1**. Docker downloads a Linux image with everything required to run ASP.NET Core already installed. Next, it adds an image in which to install your app. Containers are "composed" by layering one image on top of another to get all of the functionality you need. By maintaining the original ASP.NET Core image without modification, Docker won't have to download that container again the next time you build any project based on that image.

The very first build may take several minutes. Be patient. Eventually, you'll get a browser showing some JSON, the same output as the project template for ASP.NET Web API project. Leave the app running, open the Values controller in the Solution Explorer and put a breakpoint inside the initial Get() method, and then refresh your browser. Congratulations! You're debugging an application that's running on Linux in a Docker container.

Next, you'll upgrade from .NET Core 1.0.1 to the latest version (currently 1.1.0). The process is still a bit messy, but will probably improve before VS 2017 is in RTM. A lot of the work you'll want to do requires the latest versions, so it's worth the time to learn how to upgrade. Right-click on the project file and choose **Manage NuGet Packages**. Select the **Updates** tab and update all of the packages. You may have to do more than one round of updates to get them all updated. Right-click on the project again, choose **Properties** to open the project properties page and change the Core framework version from 1.0 to 1.1. Finally, open **Dockerfile** (no extension) in the **Solution Items** folder for editing. The top line says FROM microsoft/aspnetcore:1.0.1. This is the base image that Docker downloads to create your container (before your app is layered on). You can find information about the latest containers posted by Microsoft at https://hub.docker.com/r/microsoft/aspnetcore. Edit the line to read: FROM microsoft/aspnetcore:1.1.0. That's the latest version as I write this. Rebuild as a .NET Core 1.1 app running on a Linux image with .NET Core 1.1 pre-installed.

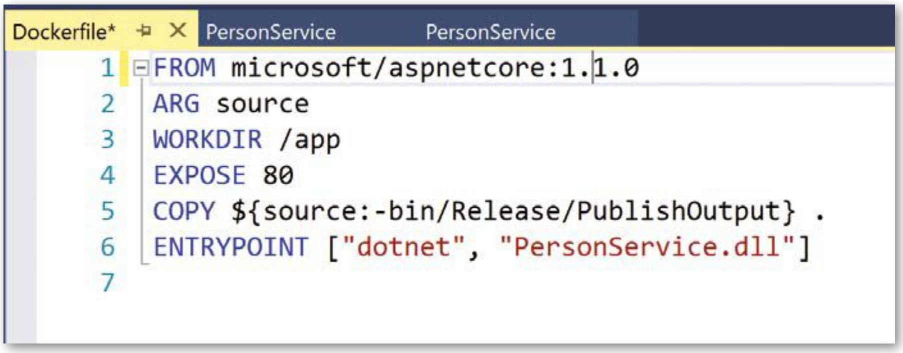
Create a Database

This article isn't about how to create databases or write Web API services, but let's do something a little more interesting than return some static text. Create a new SQL Server database (use SQL Server 2016 if you plan to move this to Azure), name it as you like, and then run the script included with the download for this article or manually create a new table with the columns and then add a few rows of test data, as in **Table 1**.

Because you'll be hitting the database from Linux, you won't be able to use a trusted connection (Windows credentials) to connect to the database, so you'll create a login and user named devuser with password P@ssw0rd (or choose your own user name and password, but remember that if you push the database up to Azure later in this article, the password will have to meet Azure's strict password requirements) and give that user permission to read and write all tables. If you haven't downloaded the script, you'll need to add these manually using SSMS. Test that you can log in as devuser and read and write the Person table. This is important, because if you can't connect with SSMS, your service won't be able to connect either.

Install Entity Framework Core

Next, you'll use Entity Framework Core (EF Core) to talk to the database. Back in Visual Studio, right-click on the project and select Manage NuGet Packages. Choose the Browse tab and search for and install the following:



```
FROM microsoft/aspnetcore:1.1.0
ARG source
WORKDIR /app
EXPOSE 80
COPY ${source:-bin/Release/PublishOutput} .
ENTRYPOINT ["dotnet", "PersonService.dll"]
```

Figure 2: Enable Docker container support

Id	UNIUQEIDENTIFIER	NOT NULL	DEFAULT: newid()
FirstName	NVARCHAR(50)	NOT NULL	DEFAULT: ''
LastName	NVARCHAR(50)	NOT NULL	DEFAULT: ''
oEmail	NVARCHAR(100)	NOT NULL	DEFAULT: ''
Phone	NVARCHAR(50)	NOT NULL	DEFAULT: ''

Table 1: The test data

1. Microsoft.EntityFrameworkCore
2. Microsoft.EntityFrameworkCore.SqlServer
3. Microsoft.EntityFrameworkCore.SqlServer.Design
4. Microsoft.EntityFrameworkCore.Tools

For the last one, you may have to check the Include Prereleases checkbox on the search screen. As I'm writing this article, there's still not a released version of this package. This package adds some development-time capabilities to the NuGet Package Manager Console, including the ability to generate Code First EF classes from your existing database. Yes, Code First migrations also work if you prefer that approach, but both features are still in preview. You can find more information about the tools at http://docs.microsoft.com/en-us/ef/core/miscellaneous/cli/powershell.

Under the Tools menu in Visual Studio, select **NuGet Package Manager, Package Manager Console**. Type in the following command at the prompt (substituting the name of your server, database and login credentials as necessary).

```
scaffold-dbcontext "Server=mycomputer;
Database=AzureDockerServices; User Id=devuser;
Password=P@ssw0rd"
Microsoft.EntityFrameworkCore.SqlServer
-OutputDir EFModels
```

This command adds a folder named **EFModels** to your project and then generates a context class and a class for each table in that folder. EF Core is very much a V1 product. It's not nearly as complete and robust as its predecessors, however all of the basics are there to make most common tasks possible and future versions promise to be better than the EF we use today. If you need more complex interactions, .NET Core has many database connectors, including one for SQL Server that you can use to perform any activity still out of the reach of EF Core.

Create a Web API Service

Right-click on the Controllers folder, select **Add, New Item.** Under ASP.NET Core, Web, ASP.NET choose **Web API Controller Class** and name it PersonController.cs. You won't be using the sample methods, so delete them to prevent a conflict. Paste the following method into the class.

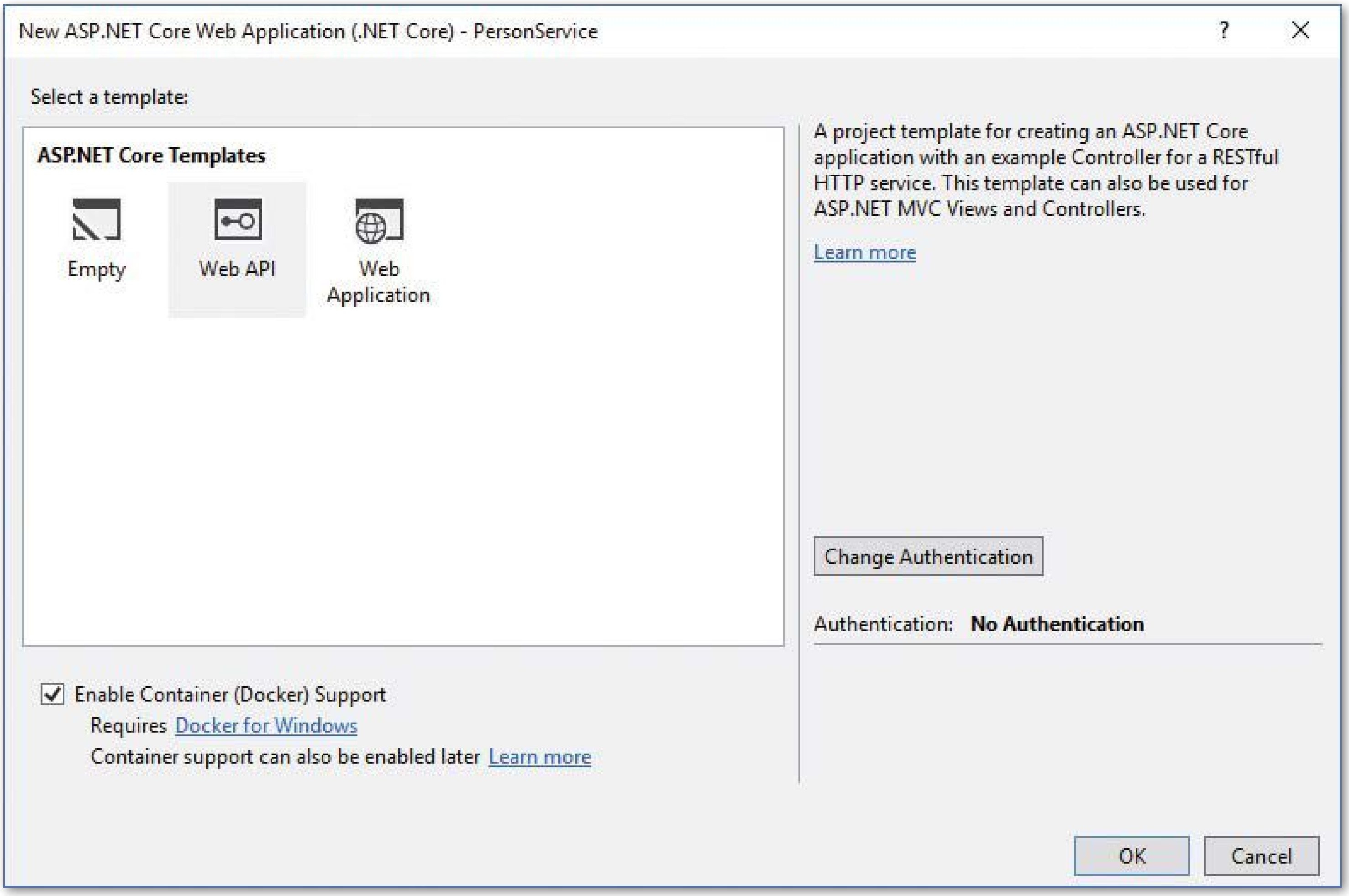
```
[Route("{searchText}")]
[HttpGet]
public List<Person> SearchPeople(string searchText)
{
  using (var context = new
AzureDockerServicesContext())
  {
    var people = context.Person
       .Where(p => p.FirstName.StartsWith(searchText)
           || p.LastName.StartsWith(searchText)
           || p.Email.StartsWith(searchText)
           || p.Phone.StartsWith(searchText))
       .ToList();

    return people;
  }
}
```

Run the app again. When the browser appears, replace **/api/values** in the URL with **/api/person/m**. This calls your method in the person controller and returns all records where the first name, last name, email, or phone number starts with the letter **m**. If you run into a SQL error, edit AzuredickerServicesContext.cs and try using your IP address instead of your computer name. The Linux computer running in Docker may not be able to do a DNS lookup on your computer name if you're not connected to a network. I've run into this a few times while travelling with my laptop. I found that 127.0.0.1 doesn't work either. Run ipconfig at a command prompt and use the IP4 address listed there. You might have noticed that the code you wrote would work equally well in a standard ASP.NET MVC project.

Push the Database to Azure

Now that you have a working service on your dev computers, let's deploy the whole thing to Azure. Start by deploying the SQL Database. I'm assuming that you have an Azure account. If not, you can create a free trial account and use that for testing. Use the Azure portal to create a Create a SQL Server if you don't have a server set up already. Make sure that you can connect to the server in Azure with SSMS. Add the devuser login to the server so that when you push the database up to Azure, you'll be able to push the devuser user as part of the database.

Open SSMS 2016 login to your local database making sure you're logged in as an admin and not as devuser.


New ASP.NET Core Web Application (.NET Core) - PersonService

Select a template:

ASP.NET Core Templates

A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET MVC Views and Controllers.

Authentication: **No Authentication**

[✓] Enable Container (Docker) Support

Requires Docker for Windows

Container support can also be enabled later Learn more

Figure 3: Edit DockerFile to use the aspnetcore:1.1.0 image.

- 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

Right-click on the AzureDockerServices database. Choose **Tasks, Deploy Database to Microsoft Azure SQL Database**. Log into your server in Azure. You can accept the defaults to keep the database name and the Basic edition, which has a 2GB size limit. Cost for this database is about $5/month if you keep it for the whole month. Finish the wizard to push your database up to Azure. Using SSMS, log in to the Azure server as devuser, and make sure that you can read and write the person table.

Open AzuredockerCoreServicesContext.cs in the EFModels folder and change the name of the server to point to the database in Azure. Test the service to ensure that it's still working properly.

Push the Container to Azure

The final step is to publish the Docker container with your services app to Azure. For now, you're just going to publish a single copy of the Linux container. If it works there, the same container can be deployed with Docker Swarm or some other container orchestration product for more advanced deployment and scalability scenarios.

Right-click on the project and choose **Publish**. On the Publish tab, click **Create** to create a new deployment profile. Choose **Azure App Service Linux (Preview)**. Change the Web App Name if you like, choose the Azure subscription you want to use, and create a new Resource Group. I named mine Docker-WestUS-ResourceGroup so I can easily tell which data center region I'm using. Create a new App Service Plan. I named mine Docker-WestUS-Plan. Next, create a new Container Registry. I named mine DockerWestUSRegistry. Container Registry names cannot contain dashes and the error message is misleading. Note that currently only West Europe, Southeast Asia, and WestUS regions offer Linux App Services. Your Resource Group, App Service Plan, and Container Registry must all be in the same Azure region. Don't use an existing Resource Group unless you know for sure it's in the same region as the plan and registry.

Unlike deploying an ASP.NET MVC and/or Web API project, publishing doesn't automatically start up a browser for you to go try it out.

Once you publish, the app builds and deploys. The deployment in the Output window happens pretty quickly, and then a command window opens and you'll see your image being pushed up to Azure. In Docker lingo, an image is a file and a container is a running instance of that file. Images are stored in repositories. When you want create a new container, you make an image and deploy a copy of the image to a repository. When you want to spin up a new container, you get a copy of the image from the repository and start it up on some hardware.

Unlike deploying an ASP.NET MVC and/or Web API project, publishing doesn't automatically start up a browser for you to go try it out. A link to the URL shows up in the Web Publish Activity window, however. If you can't find it, log into your Azure portal, open up Web Apps, find your new app, and use the link on the overview page. Initially, you get a 404 error. Don't forget that you have to add **/api/person/m** to the end of the URL to hit the service.

Summary

Well done! You created a Linux app that hits SQL Server and ran it not only in your desktop environment but also in the cloud. You're now a certified services developer for Docker.

Services are the first things you expect to see built with .NET Core. Even though it's a brand new product and will grow more powerful with each new version, there's already enough in this portion of .NET Core to make it worth giving a serious look. Websites and other types of projects are sure to follow, but they're not as far along as services. Because .NET Core can run on a variety of platforms, it makes sense to take a serious look at Linux. And because Docker now runs on Windows, it makes sense to take a serious look at Docker. These are hot job skills and they're getting a lot of press for good reason. As I review this before sending it off to my editor, a pop-up just notified me that a new version of Docker has just been released, version 1.13, the one with support for Windows containers is now in RTM. That's an article for another day. It's a fast-paced world and moving faster all the time, but it's not a big stretch for a C# developer to hop on and see what all the fuss is about.

Mike Yeager

CODE

(Continued from 74)

(For the record, it was the most gold-plated About box you've ever seen. Scrolling text, a video Easter egg—you name it, our About box had it. It was a virtual monument to one man's ability to put every possible C++/Windows thing into a single, solitary dialog box of no real importance. It was, by far, the single most impressive thing in our application that nobody ever looked at.)

It's nice that Mike could code, but we didn't need him to be the technical expert on the team. We had a Technical Lead, and he made pretty good decisions. Mike was, to be sure, thoroughly proficient in C++ and Windows, but the truth is that the rest of the team was (probably) leaving him behind in raw technical knowledge, particularly as time went by and we learned more on a day-to-day basis that he didn't get the chance to. The truth was that we didn't need him to be the technical lead. We needed him to be our manager, making sure our desktops were working, the team room was ours (which it wasn't when I first started), and so on.

The more hours that a manager is spending heads-down in the IDE, the less that manager is available to resolve conflicts, make decisions, or erase obstacles that only somebody higher up the food chain can. Like the time the VP of Operations thought an internal outage was due to something somebody on our team had done—Mike went to bat for us, insisting that nothing we were doing would've caused the outage (which was true). Eventually it was "discovered" that the fault had originated from within the Operations group. (And the VP's voice, screaming at the top of his lungs at his Operations team for their mistake, could be heard throughout almost the entire building—which reminds me, never ever be *that* kind of manager.)

Look at it this way: if a manager is working an hour on code, it's one man-hour of code contributed to the project. But if the manager can unblock the team from an obstacle, it's a half-dozen or more man-hours of (perhaps more productive) development time that's now added. The manager becomes, as the military calls it, a *force multiplier*; their actions essentially multiply the effectiveness of the rest of the group, carrying far greater weight than their actions alone ever could.

And hey, if the net effect is to make the team better, does it really matter all that much if you're the one writing the code?

Summary

It's helpful, certainly, for managers to know how to code—having the ability to understand the issues that your team faces, and being able to weigh in with suggestions or ideas, even if they are more vague or abstract than concrete and specific, is a valuable thing. But a manager has to recognize that this is not their principal contribution to the team; they have a different title for a reason. Despite how the manager may feel, theirs is the role of facilitator and, sometimes, coach.

Which means it's maybe OK for managers to code the About box, even if it's one of those gold-plated Cadillac Escalade versions of an About box. But I wouldn't feel comfortable with them doing much more than that; they've got much better things to do.

Ted Neward

CODE COMPILERS

CODE

Mar/Apr 2017
Volume 18 Issue 2

Group Publisher
Markus Egger

Associate Publisher
Rick Strahl

Editor-in-Chief
Rod Paddock

Managing Editor
Ellen Whitney

Content Editor
Melanie Spiller

Writers In This Issue
Mohammad Azam, Jason Bender
Sahil Malik, Ted Neward
Jeffrey Palermo, John V. Petersen
Rachel Reese, Paul D. Sheriff
Chris 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 026
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
832-717-4445 ext 028
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, email subscriptions@codemag.com
or contact customer service at
832-717-4445 ext 028.

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 Coding Management

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. "So you've stopped coding, then?" Several times over the last few weeks I've been on the receiving end of this question. It usually comes from somebody who's known me (or known of me) and they find out that my current role is as Director of Developer Relations for a hockey-stick-scaling startup called Smartsheet (http://www.smartsheet.com –check 'em out, get a developer license, and make my CEO happy!). I got the same question several times in the prior two years too, when I was the CTO of the consulting company startup. There's this unwritten rule, somehow, that if "manager" or "director" or "VP" appears anywhere in your title, you've clearly stepped over that "not a coder" line.

And, strangely, I find myself growing a touch defensive about the question, like somehow it's an admission of failure or something, instead of the natural progression of the developer to senior developer to architect or team lead to...

Management.

It's a subject that's on my mind, too, because Smartsheet is conducting its annual employee review process, and one of the questions asked is, "Is your manager technically skilled enough to support you in your position?" At Smartsheet, almost anyone in any management capacity is expected to be hands-on coding at least some non-trivial percentage of the time, and, not surprisingly, almost all of them do.

And yet...should they? Should they have to? Should they want to?

What Managers Do

It's not an uncommon question: What, exactly, do managers do for us? Certainly, recent industry experiments would seem to suggest that a manager-free lifestyle is not a terrible thing to live. After all, with no managers, there's no status reports, and no meetings to go to, and no "Yeahhhhhh...if you could have those TPS reports by Friday, that'd be great" that developers need to deal with.

But, like the myth of the open office plan, the myth of the manager-free lifestyle is slowly eroding. Google, in particular, is discovering that a workplace where developers simply roam where they will can yield some impressive experiments and projects, but doesn't always yield what the customer (or the project manager, or upper management, or...) asked for. Part of that is because the developer doesn't have easy access to the people asking for the project in the first place—the customers are on a different floor, different building, different company, or different continent, and as a result, the developer ends up building what the developer thinks the customer asked for, rather than what the customer actually asked for.

And, worse, sometimes the developers know that they don't know what the customer wants, but without a manager, they have limited-to-no ability to navigate across the org chart to find out. This leaves them with the only real option left to them at that point: guessing.

This, in part, is what managers are supposed to do: When an issue or obstacle arises that keeps people from getting their jobs done, it's the manager's responsibility to make that obstacle or issue go away. Sometimes those obstacles are internal in nature, such as a logjam among the developers around what language to use or framework to adopt or methodology to use or some other technical decision needs to be made and the team is stuck. Sometimes those obstacles are external in nature, such as getting those-who-are-asking-for-the-software to sit in a room with those-who-are-building-the-software so they can build the software to do what it actually needs to do. And sometimes the obstacles are physical in nature, such as when a developer needs a 34" monitor to...er... um...well, I'm sure there's some kind of good reason that a developer needs a monitor that big. Or to manage the physical grow-out of the company so that developers aren't crammed four-to-an-office. Or any of a thousand other things that, when not properly handled, send developers' productivity plummeting.

This isn't to argue that all managers are good managers. Every developer knows, through our collective unconscious, of managers who put up more obstacles than they eliminate. But to paint all managers with that brush is to commit the same sin that managers do when they paint all developers with the "prima donna" brush. Or the "they're all bespectacled nerds" brush. Or the "400-pound hacker sleeping in his parents' basement," as a particular political candidate put it.

Famed ScrumMaster Trainer, and Planning Poker aficionado Mike Cohn was, to my great delight, one of my first managers when I began programming. (Ask him sometime if you see him at a conference—he's got some stories). He once described his job as "being the umbrella over the development team, to keep all the management off of them so they could actually get work done." And, largely, that's what he did, and we were able to get some stuff done.

What Managers Don't Need to Do

Code.

Seriously. They don't need to be coding. It's nice that they can, of course, because certain situations call for a degree of technical expertise as part of the decision-making process. And some choose to do so because they came from these same roots that you and I do, and see coding as an enjoyable activity to which they can return at times, like salmon returning to their spawning pool periodically to perpetuate the species (or, in this case, to recharge their batteries so they can go out and fight the managerial battles).

As a matter of fact, I'll never forget one day when Mike came into our team room and asked a buddy of mine if he was the one to fix bug #453. My buddy and I looked at each other, then both dived into the bug-tracking system to figure out which bug #453 was. Turned out, it was a bug in the About box (this was back in the mid-90s and we were working on a desktop system. Don't judge), and Ron had, in fact, fixed it. In a very soft voice, Mike said, "I need you to back that fix out." Now thoroughly confused, Ron and I just looked at each other until Mike explained further. "I spend close to forty hours a week slogging through management meetings and reports so you guys don't have to, and every Friday, after lunch, I close the door to my office and work on the code. The About box is the only thing I'll let myself work on, and I'll be *damned* if I let you guys take that away from me!"

(Continued on page 73)

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

Affordable High-Tech Digital Escrow

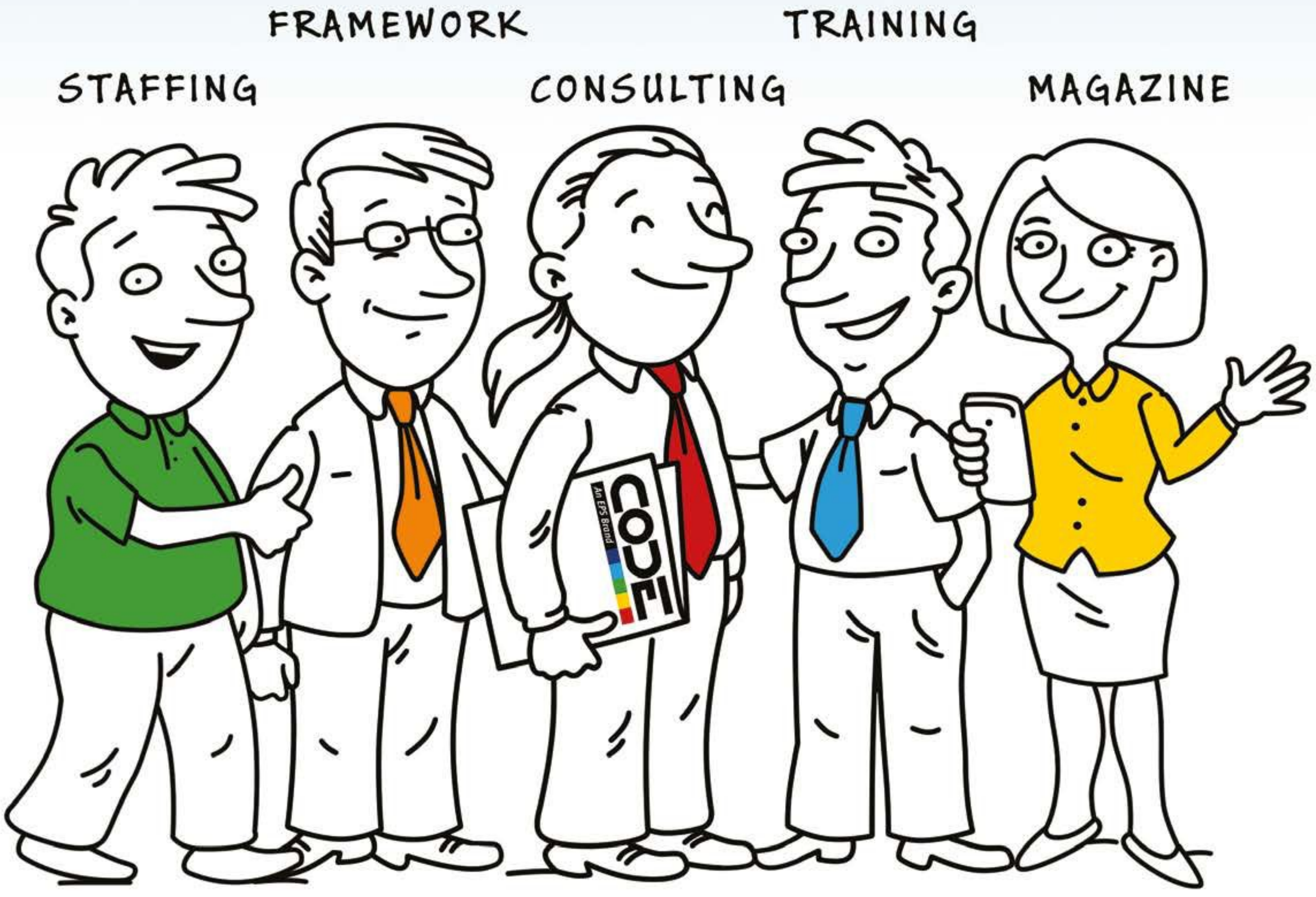
Tower 48 is the most advanced and affordable digital escrow solution available. Designed and built specifically for software and other digital assets, Tower 48 makes escrow inexpensive and hassle free. Better yet, as a vendor, you can turn escrow into a service you offer to your customers and create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

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