THE MICROSOFT JOURNAL FOR DEVELOPERS

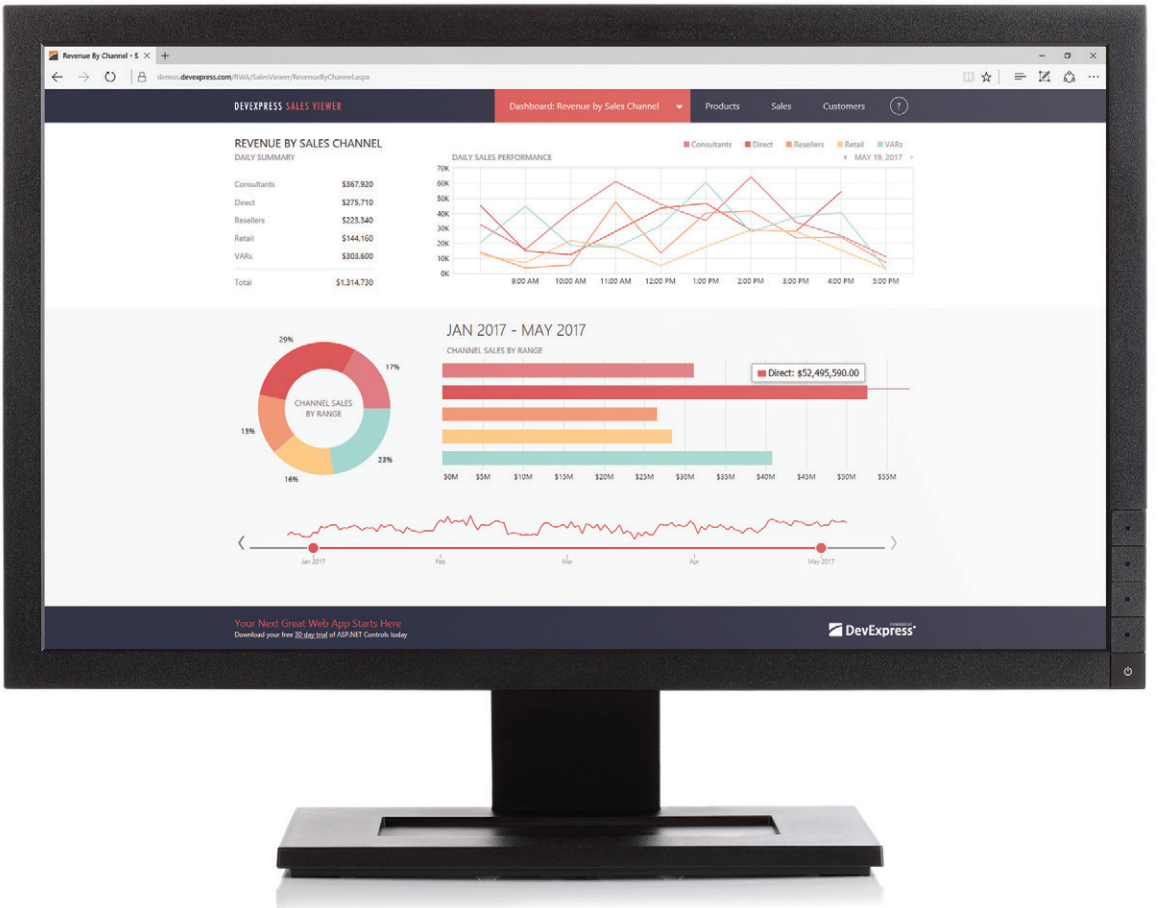
FEBRUARY 2018 VOL 33 NO 2

msdn magazine

DevExpress®

Your Next Great Web App Starts Here

From apps that replicate the look and feel of Microsoft Office® 365, to high-impact decision support systems for your enterprise, DevExpress Web Controls for ASP.NET will help you build your best, without limits or compromise.



Download your free 30-day trial
and experience the DevExpress difference today.

devexpress.com/try

THE MICROSOFT JOURNAL FOR DEVELOPERS

FEBRUARY 2018 VOL 33 NO 2

msdn magazine

COLUMNS

Microsoft

INFRAGISTICS

Infragistics Ultimate 17.2

Productivity Tools & Fast Performing UI Controls for Quickly Building Web, Desktop, & Mobile Apps

Includes 100+ beautifully styled, high-performance grids, charts & other UI controls, plus visual configuration tooling, rapid prototyping, and usability testing.

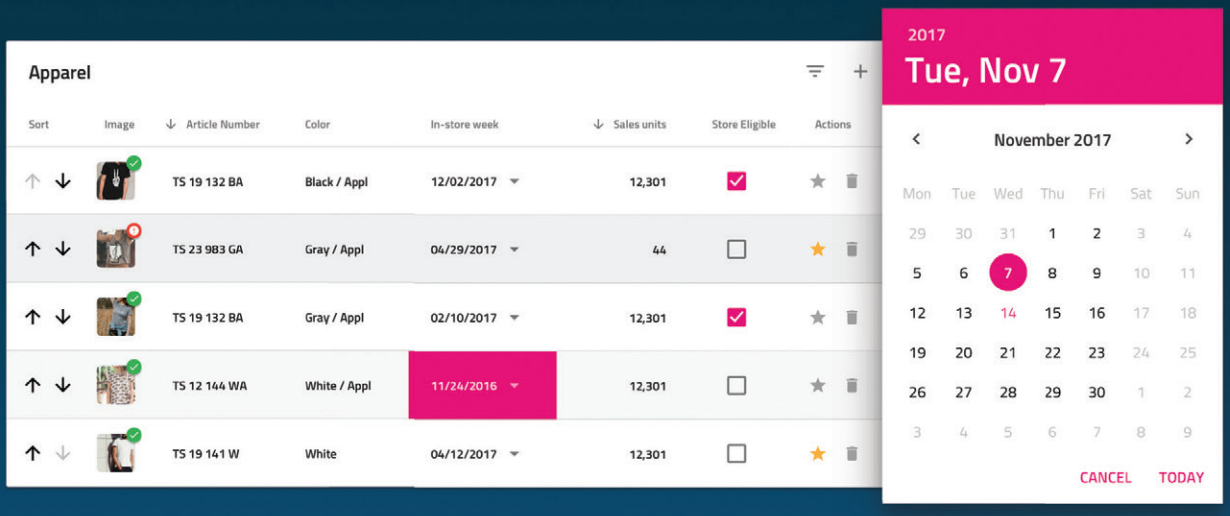
Angular | JavaScript / HTML5 | ASP.NET | Windows Forms | WPF | Xamarin

Download a free trial at
Infragistics.com/Ultimate

Featuring

Ignite UI

A complete UI component library for building high-performance, data rich web applications



- Create beautiful, touch-first, responsive desktop & mobile wep apps with over 100 JavaScript / HTML5, MVC & **Angular components**.
- Our easy to use Angular components have no 3rd party dependencies, a tiny footprint, and easy-to-use API.
- The Ignite UI **Angular Data Grid** enables you to quickly bind data with little coding - including features like sorting, filtering, paging, movable columns, templates and more!
- Speed up development time with responsive layout, powerful data binding, cross-browser compatibility, WYSIWYG page design, & built-in-themes.

Download a free trial of Ignite UI at: **Infragistics.com/Ignite-ui**

To speak with our sales team or request a product demo call: 1.800.321.8588

msdn magazine

FEBRUARY 2018 VOLUME 33 NUMBER 2

ID STATEMENT MSDN Magazine (ISSN 1528-4859) is published 13 times a year, monthly with a special issue in November by 1105 Media, Inc., 9201 Oakdale Avenue, Ste. 101, Chatsworth, CA 91311. Periodicals postage paid at Chatsworth, CA 91311-9998, and at additional mailing offices. Annual subscription rates payable in US funds are: U.S. $35.00, International $60.00. Annual digital subscription rates payable in U.S. funds are: U.S. $25.00, International $25.00. Single copies/back issues: U.S. $10, all others $12. Send orders with payment to: MSDN Magazine, P.O. Box 3167, Carol Stream, IL 60132, email MSDNmag@1105service.com or call (847) 763-9560. POSTMASTER: Send address changes to MSDN Magazine, P.O. Box 2166, Skokie, IL 60076. Canada Publications Mail Agreement No: 40612608. Return Undeliverable Canadian Addresses to Circulation Dept. or XPO Returns: P.O. Box 201, Richmond Hill, ON L4B 4R5, Canada.

CORPORATE ADDRESS 1105 Media, 9201 Oakdale Ave. Ste 101, Chatsworth, CA 91311 1105media.com

MEDIA KITS Direct your Media Kit requests to Chief Revenue Officer Dan LaBianca, 972-687-6702 (phone), 972-687-6799 (fax), dlabianca@1105media.com

REPRINTS For single article reprints (in minimum quantities of 250-500), e-prints, plaques and posters contact: PARS International
Phone: 212-221-9595
E-mail: 1105reprints@parsintl.com
Web: 1105Reprints.com

LIST RENTAL This publication's subscriber list, as well as other lists from 1105 Media, Inc., is available for rental. For more information, please contact our list manager, Jane Long, Merit Direct. Phone: 913-685-1301;
E-mail: jlong@meritdirect.com;
Web: meritdirect.com/1105

Reaching the Staff

Staff may be reached via e-mail, telephone, fax, or mail.

E-mail: To e-mail any member of the staff, please use the following form: FirstinitialLastname@1105media.com

Irvine Office (weekdays, 9:00 a.m. – 5:00 p.m. PT)
Telephone 949-265-1520; Fax 949-265-1528
4 Venture, Suite 150, Irvine, CA 92618

Corporate Office (weekdays, 8:30 a.m. – 5:30 p.m. PT)
Telephone 818-814-5200; Fax 818-734-1522
9201 Oakdale Avenue, Suite 101, Chatsworth, CA 91311

The opinions expressed within the articles and other contentsherein do not necessarily express those of the publisher.

Microsoft

BPA WORLDWIDE

1906 100 2006 AMERICAN BUSINESS MEDIA

WPA Western Publications Association

Build Better Apps with LEADTOOLS® V20

Target every major platform with new libraries for .NET Standard, .NET Core, and Xamarin. Advanced technologies, including OCR, Barcode, PDF, DICOM, PACS, Multimedia, and more, are just a few lines of code away!

SDKs for DOCUMENT

SDKs for MEDICAL

SDKs for MULTIMEDIA

SDKs for IMAGING

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

DOWNLOAD OUR 60 DAY EVALUATION
WWW.LEADTOOLS.COM

LEAD TECHNOLOGIES INCORPORATED

SALES@LEADTOOLS.COM
800.637.1840

MICHAEL DESMOND

Groundhog Day

Groundhog Day has to be my favorite holiday. On the 2nd of each February, the citizens of tiny Punxsutawney, Pa., gather at Gobblers Knob to bear witness to a celebrated rodent predicting the weather. If the groundhog emerges and sees its shadow, the legend goes, the country will experience another six weeks of winter. If not, the forecast is for an early spring.

I mean, you can't make this stuff up.

The day is rooted in ancient celebrations at the midpoint between the winter solstice and spring equinox. On the Gaelic holiday of Imbolc a mystical hag called the *Cailleach* was said to gather firewood to last her the rest of winter. If she wanted winter to last, she conjured sunny weather for her gathering, while cloudy or poor weather meant the Cailleach was asleep and winter nearly over. All of which *I think* explains why our magical, weather-controlling groundhog calls for six weeks of winter if the sun shines on Feb. 2.

On the 2nd of each February, the citizens of tiny Punxsutawney, Pa., gather at Gobblers Knob to bear witness to a celebrated rodent predicting the weather.

Whatever the case, predictions are in the air. So I turned to our intrepid columnists for their takes on the future, and here's what they said.

No surprise, artificial intelligence (AI) is on tap. **Frank La Vigne**, who writes the Artificially Intelligent column, says that within two years AI will be a "mandatory skill set" for mainstream developers, much the way mobile development emerged over the past decade.

Test Run author **James McCaffrey** expects enterprise developers to take over writing prediction code the same way they've taken on other parts of the pipeline, such as design, implementation and test. And he says that this code, currently a rarity in enterprise production software systems, soon won't be. "By January 2019, the use of neural network classification and regression prediction code (generated using the Microsoft CNTK library, or inserted via Azure Services) in production systems will be far more common."

David Platt (of Don't Get Me Started fame) thinks AI is doomed to meet its match when it "runs smack into Natural Stupidity™," while Essential .NET author **Mark Michaelis** helpfully points out that I'll likely use AI to generate my prognostications for next year's Groundhog Day column.

On the security front, Spectre and Meltdown will echo far into 2018 and 2019, says **Ted Neward**, author of The Working Programmer. "These are the first widely publicized 'side-channel' attacks to make the evening news. Even if they never actually result in actionable attacks in the wild, the patches are already confirmed as reducing the performance of the chips they protect."

Julie Lerman, author of the Data Points column, sees changes coming for microservices design, which she says could benefit from proven Domain-Driven Design (DDD) concepts. "DDD provides guidance on defining the boundaries of your processes using bounded contexts and maintaining separate databases per each bounded context, just as we do with microservices," she says. "It has also solved the constant question about how to share data between those contexts."

Two columnists address cryptocurrencies and blockchains. Michaelis says cryptocurrencies will continue to increase in value, not necessarily because of any new innovation, but because people can't resist a get-rich scheme. But La Vigne is bullish on blockchains. "I do think we will see some creative uses for blockchains, especially in the supply chain space to enforce ethical standards and food safety laws."

As for Platt, he looks to the day when Amazon buys Walmart and converts its stores into delivery drone bases. He's also waiting to see which major university will be first to replace its science department with StackOverflow.

Visit us at msdn.microsoft.com/magazine. Questions, comments or suggestions for *MSDN Magazine*? Send them to the editor: mmeditor@microsoft.com.

DevExpress®

DevExpress Spreadsheet for WPF & WinForms

with Chart, Pivot Table and Cell Editor support

Your users already know Microsoft® Excel® and with the new capabilities we've introduced in our Spreadsheet Control, you can create solutions that utilize a "spreadsheet-first" UX for a broad range of use-case scenarios. Imagine sales entry forms that are an actual invoice or an inventory management document with built-in ordering and data entry options.

Give our Spreadsheet a try today and see how easy it is to build Windows® apps that amaze.

Free 30-day trial
devexpress.com/spreadsheet

#UseTheBest

JULIE LERMAN

Creating Azure Functions That Can Read from Cosmos DB with Almost No Code

In my previous two columns, I created a simple Universal Windows Platform (UWP) game, CookieBinge, that tracks the number of cookies a game player gobbles up. In the first of those columns, I showed you how to use Entity Framework Core 2 (EF Core 2) to store the game scores locally on the Windows 10 device where the game is being played. In the second column, I walked you through building an Azure Function that would take game scores sent over HTTP and store them into a Cosmos DB database. The ultimate goal is to modify the game to be able to send its scores to this Function, as well as to retrieve score data from parallel functions.

In this column, I'll show you how to create the Azure Functions that will respond to requests to retrieve two sets of score data: the top five scores for one user across all of that user's devices, and the top five scores of all users playing the CookieBinge game around the world. Next time, in the final column in the series, I'll integrate the new Azure Functions into the CookieBinge UWP app to allow users to share and compare their scores.

The Azure Function I created in the previous article responds to an HTTP call and, using the Function's integrations, is able to push the data from the HTTP request into the Cosmos DB database without me having to write any data access code. **Figure 1** displays one of the documents stored in the Binges collection of the CookieBinge database. The first seven properties are what I stored ("logged" through "worthit") and the rest are metadata created by Cosmos DB when the record was inserted.

For the next two functions, I'll again take advantage of Azure Function integrations to interact with the same Cosmos DB database. The only code required for the database interaction this time will be some SQL that represents the necessary query string, and this gets built into the integrations, not the function's code. Both functions will be triggered by HTTP calls. The first will take in the ID of the user making the request and then return the top five scores the user logged across all of the devices on which they played the game. The second won't take any input; it will simply return the top five scores of all gamers from around the world.

I'll walk you through building the first of these Functions using the Azure portal workflow, adding on to the lessons of the previous column. Walking through the first of these functions should help you comprehend how all of the puzzle pieces fit together. I'll then show you a shortcut when it's time to build the second. Remember that it's also possible to create Azure Functions with the tooling built into Visual Studio 2017, as well as by using the Functions command-line interface (CLI) plus Visual Studio Code.

Code download available at msdn.com/magazine/0218magcode.

Creating a Function to Return a User's Top Scores

Let's return now to the CookieBinge Azure Functions app in the Azure portal and add a new Function to the app. Recall that when you hover over the Functions group header under the function app's name, you'll see a plus sign on which you can click. When you do, you'll be prompted to choose a template for the new function and, as with the first function you created, it will be a C# HTTP trigger. Name the new function GetUserScores, change the authorization level to Anonymous, then click Create.

In the previous article, I wired up an output integration to the CookieBinge Cosmos DB database, which caused the results of the function to be stored into the database.

Before modifying the code in the run.csx file, I'll build the function's integrations. Recall that there are three types of integrations:

- Trigger: There can only be a single trigger and in this case, it's the HTTP request to the Function.
- Input integrations: There can be multiple input integrations and these represent other data you want to be available to the Function.
- Output integrations: In the previous article, I wired up an output integration to the CookieBinge Cosmos DB database, which caused the results of the Function to be stored into the database. With this new GetUserScores function, the output will be the results of the Function sent back to the requestor as an HTTP response.

I'll wire up the input integration to retrieve data from the Cosmos DB database where the StoreScores Function from last month's article stored them. This input was a confusing concept for

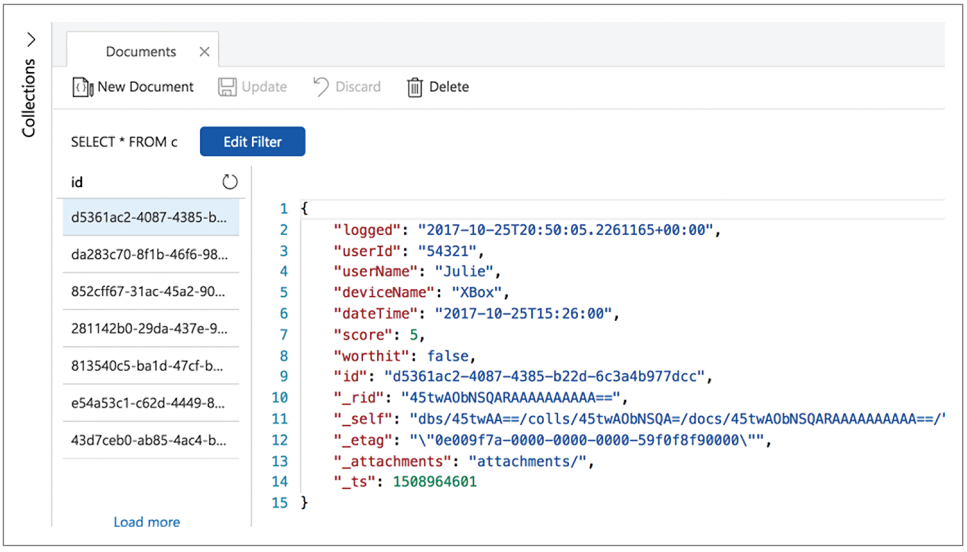


Figure 1 **A Record from the CookieBinge Binges Collection**

me at first. I assumed I'd have to write some code in the Function to connect to that database and query for the users' scores. But the input integration is pretty cool. I can define it with the relevant SQL for querying the Cosmos DB database and let it know to use the UserId value that was sent in with the HTTP request. And Azure Functions takes care of the rest—connecting to the database, executing the query and handing me the query results to work with in the function code. Yes, very cool indeed! Let's do it.

Creating an Input Integration to Read from Cosmos DB Using HTTP Request Data

Start by clicking New Input in the Integrations panel, then from the list of input types presented, choose Cosmos DB and click the Select button below it. This results in the Cosmos DB input form, its fields populated with default values. Modify the values as shown in **Figure 2**.

The values for the Database name, Collection Name and Azure Cosmos DB account connection should match the ones you used for creating the StoreScores function's output integration. In my case, that's CookieBinge, Binges and datapointscosmosdb_DOCUMENTDB. I've named my parameter "documents," which I'll use in my function's code. Document ID is left blank. If you were building a function to retrieve a single Cosmos DB document by its ID, this would be a handy option. But this function will be querying based on an incoming UserId.

The SQL Query is cut off in the screenshot, so here it is in full. Keep in mind that you need to add it as a single line, though here I've used line returns to make it easier to read:

```
SELECT  TOP 5 c.score,c.worthit,c.
deviceName,c.dateTime
FROM c
WHERE c.userId={userId}
ORDER by c.score DESC
```

Azure Cosmos DB has a SQL query syntax that makes it nicely familiar to work with.

Notice the placeholder for the userID variable, {userId}. I'll use this variable to wire up the trigger and this input integration. First, I need to save the settings of this input. Then I'll need to create a binding parameter to bind the trigger to the input. This can be done in a way that was not immediately obvious to me. In fact, before I got it set up correctly, I kept getting an error about the binding parameter being missing and it took some effort putting together what I was learning from the docs, Web search results and some experimentation before I got it right. Part of the problem was my lack of understanding, although once I found the solution, the connections made sense to me. Hopefully you'll see the light more quickly because I'm about to spoon-feed the solution to you!

Azure Cosmos DB has a SQL query syntax that makes it nicely familiar to work with.

Select the HTTP (req) trigger and specify a route template for the trigger. The template should first provide a name for the route, and then specify the binding variable, userId.

The route doesn't have to match the name of the function, but it's probably clearer if it does. I named mine GetUserScores/{userId}, as shown in **Figure 3**. Don't forget to save your changes.

The output integration can stay at its default, which is an HTTP response.

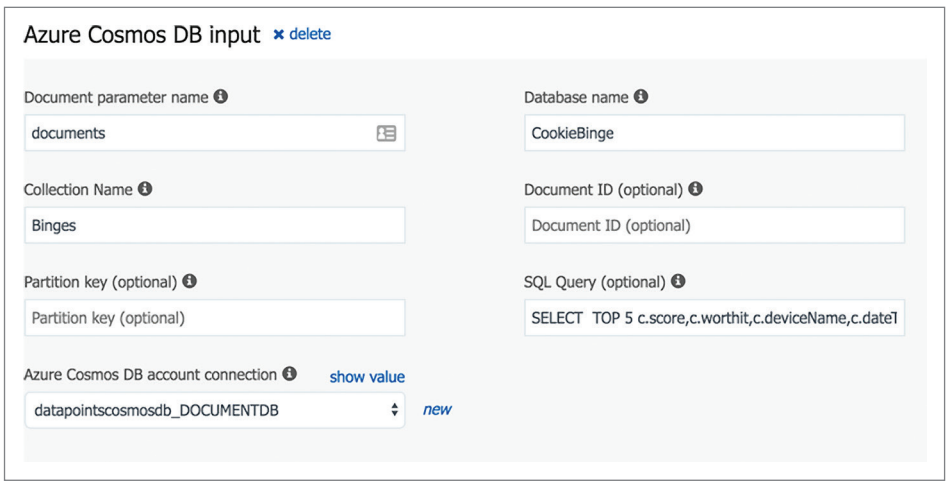


Figure 2 **The Azure Cosmos DB Input Integration Settings**

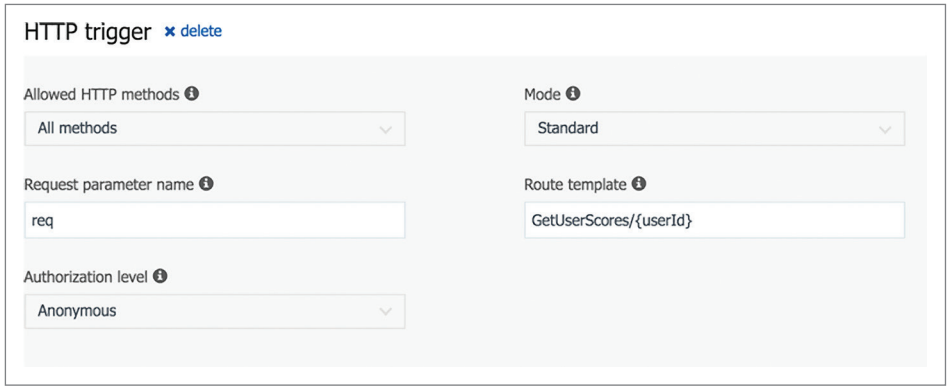


Figure 3 **The HTTP Trigger Integration Settings**

Now it's time to return to the function code, which you can do by clicking on the GetUserScores Function. This will display the run.csx code, which is still in its default state.

Writing the Function Code to Respond to the Integrations

Consider the SQL for the integration input, which retrieves four columns: score, worthit, deviceName and dateTime. You could create a class to match each type (as I did in the StoreScores Function of the previous article), or just tell the function that the input Function will be passing in a dynamic type. That's what my code does. **Figure 4** shows the entire function in the run.csx file.

In its signature, the Run method acknowledges that the first parameter is a string named userId that comes in via the route specified in the trigger settings. After the userId, the parameters expect the HttpRequestMessage (req) and then the IEnumerable containing the documents. Behind the scenes, the Function will take the userId that came in through the trigger's route and pass it on to the input integration, which will execute the query (using the userId) and populate the documents variable with the query results. After this, the Function will finally call the Run method's logic that I've written.

The code first checks that the documents were, indeed, passed in to the method. I output a message to my log with the count of those documents and then simply pass them back in the HTTP response. If I weren't doing any checks and balances, the method would still work in this simpler format:

```
public static HttpResponseMessage Run(string userId,
              HttpRequestMessage req, IEnumerable<dynamic> documents)
{
  return req.CreateResponse(documents);
}
```

In other words, for this entire Function to receive a userID and return that user's top five scores from the database takes just a single line of code that's no more than "send back the documents that came in from the input integration."

The Azure Function's integrations have really done all the work for me. I am truly impressed.

I can test this Function in the portal or in a tool like Fiddler or Postman. Testing from a browser won't work with the Function in its current state because the Function is returning dynamic objects and a browser can't send the necessary Accept header. Alternatively, you could define a class that matches the query results and use that class instead of a dynamic object. But as it is, the Function will work correctly when calling it from Fiddler, Postman or your APIs, such as the one you'll eventually use from the UWP app.

The Get Function URL link tells me I can call https://cookiebinge.azurewebsites.net/api/GetUserScores/{userId}. If the Function weren't anonymous, then I'd have to pass some credentials in, but I'm intentionally keeping it simple here.

With my userId, 54321, in the placeholder, the URL looks like this: https://cookiebinge.azurewebsites.net/api/GetUserScores/54321.

> The Azure function's integrations have really done all the work for me. I am truly impressed.

This URL returns five JSON-formatted documents with the schema shaped by the query. Here's one of the documents returned in the HTTP response, showing that when I played on my Xbox, I scarfed down 18 cookies, but wasn't so happy about my binge that day.

```
{
  "score": 18,
  "worthit": false,
  "deviceName": "XBox",
  "dateTime": "2017-11-05T15:26:00"
}
```

A Shortcut for Creating Another Fully Integrated Azure Function

Now that I have the first Function set up, it's time to create the second, which will retrieve the top five scores for all players around the globe. Rather than go through all of the settings forms again, though, I'm going to take a short cut. The settings for all of the integrations of an Azure Function are stored in the function.json file, which you can open up in the portal from the View Files pane. **Figure 5** shows the function.json for the GetUserScores Function. The bindings section wraps all of the integration settings.

Figure 4 **The Run Method of the GetUserScores Function**

```
using System.Net;
public static HttpResponseMessage Run(string userId,
              HttpRequestMessage req, IEnumerable<dynamic> documents,
TraceWriter log)
{
  if (documents != null)
  {
    log.Info($"Document Count: {documents.Count()}");
    return req.CreateResponse(HttpStatusCode.OK,documents);
  }
  else
  {
    return req.CreateResponse(HttpStatusCode.NotFound);
  }
}
```

Figure 5 **The function.json File for GetUserScores**

```
"bindings": [
  {
    "authLevel": "anonymous",
    "name": "req",
    "type": "httpTrigger",
    "direction": "in",
    "route": "GetUserScores/{userId}"
  },
  {
    "name": "$return",
    "type": "http",
    "direction": "out"
  },
  {
    "type": "documentDB",
    "name": "documents",
    "databaseName": "CookieBinge",
    "collectionName": "Binges",
    "sqlQuery": "SELECT  TOP {top} c.score,c.worthit,c.deviceName,
      c.dateTime FROM c WHERE c.userId={userId} ORDER by c.score DESC",
    "connection": "datapointscosmosdb_DOCUMENTDB",
    "direction": "in"
  }
],
"disabled": false
}
```

Figure 6 **The function.json File for GetGlobalScores**

```
{
  "bindings": [
    {
      "authLevel": "anonymous",
      "name": "req",
      "type": "httpTrigger",
      "direction": "in"
    },
    {
      "name": "$return",
      "type": "http",
      "direction": "out"
    },
    {
      "type": "documentDB",
      "name": "documents",
      "databaseName": "CookieBinge",
      "collectionName": "Binges",
      "connection": "datapointscosmosdb_DOCUMENTDB",
      "direction": "in",
      "sqlQuery": "SELECT  TOP 5 c.userName, c.score,c.worthit,c.deviceName,
        c.dateTime FROM c ORDER by c.score DESC"
    }
  ],
  "disabled": false
}
```

The first binding is the httpTrigger, noted in its type property. Note that all of the rest of its settings are described by the other properties the authLevel, name, direction and the route. Next you can see the http output binding and, finally, the input binding with all of the settings I specified in the form.

Now that you have a better understanding of all of the puzzle pieces of the Function, you don't really need to go through the forms if you don't want to. You can just create the function.json file directly and that's what I'll do for the second Function. You still need to add the new function to the Function app, so do that, again using a C# HttpTrigger template, and call it GetGlobalScores.

> Understanding and applying the bindings in their raw format can certainly save you a lot of time.

But this time, rather than going to the Integration section, open the function.json file in the View Files pane and replace it with the code in **Figure 6**. Note that the sqlQuery value is wrapped here in the listing, but that string should be on one line.

Because this request doesn't need to pass in an id, there's no route on the httpTrigger binding as with the first Function. The output binding is the same as earlier. The only difference with the input binding is that there's a different query that doesn't filter on a userId.

Understanding and applying the bindings in their raw format can certainly save you a lot of time. Taking the explicit path of first using the forms to input the settings was beneficial, however, as it helped me comprehend the settings as I was first learning about them. In fact, now that you see how the configuration is stored, you are very well prepared to build your Azure Functions in Visual Studio or Visual Studio Code, rather than in the portal.

Now for the Function code, which is nearly identical to the code for the first Function, except that it doesn't take in the userId binding as its first parameter. Here's the modified signature of the Run method:

```
public static HttpResponseMessage Run(
             HttpRequestMessage req, IEnumerable<dynamic> documents,
TraceWriter log)
```

I manually modified some of the test data in my Cosmos DB Binges collection to be sure there were a number of different userId values before running the new GetGlobalUserScores Function to validate that everything was working correctly.

Linking the UWP App with the Azure Functions

With the three Azure Functions and the Cosmos DB document database for storing and retrieving user scores in hand, the final installment of this series will return to the UWP app to integrate those Functions. Recall that the UWP app currently uses Entity Framework Core 2 to store the user scores locally onto whatever Windows 10 device where the game is being played. In the next iteration, in addition to the local storage, a user's scores (with their permission) will be sent to the StoreScores Azure Function to store in the cloud. And users will be able to retrieve a list of their own top scores across all the devices on which they've played, as well as to see the top scores of all players around the globe. The app will call the Functions created here in order to report that data to the user.

Please note that I'll probably shut down my own demo Functions at the URLs noted earlier in this article. They're hosted on my Visual Studio Subscription account, which is for testing purposes and has a spending limit. ■

Julie Lerman *is a Microsoft Regional Director, Microsoft MVP, software team coach and consultant who lives in the hills of Vermont. You can find her presenting on data access and other topics at user groups and conferences around the world. She blogs at the thedatafarm.com/blog and is the author of "Programming Entity Framework," as well as a Code First and a DbContext edition, all from O'Reilly Media. Follow her on Twitter: @julielerman and see her Pluralsight courses at juliel.me/PS-Videos.*

Thanks to the following Microsoft technical expert for reviewing this article: *Jeff Hollan*

FRANK LA VIGNE

Using Jupyter Notebooks

The Jupyter Notebook is an open source, browser-based tool that allows users to create and share documents that contain live code, visualizations and text. Jupyter Notebooks are not application development environments, per se. Rather, they provide an interactive "scratch pad" where data can be explored and experimented with. They offer a browser-based, interactive shell for various programming languages, such as Python or R, and provide data scientists and engineers a way to quickly experiment with data by providing a platform to share code, observations and visualizations. Notebooks can be run locally on a PC or in the cloud through various services, and are a great way to explore data sets and get a feel for a new programming language. For developers accustomed to a more traditional IDE, however, they can be bewildering at first.

Structure of a Jupyter Notebook

Jupyter Notebooks consist of a series of "cells" arranged in a linear sequence. Each cell can either be text or code. Text cells are formatted in MarkDown, a lightweight markup language with plain text formatting syntax. Code cells contain code in the language associated with the particular notebook. Python notebooks can execute only Python code and not R, while an R notebook can execute R and not Python.

Figure 1 shows the IntroToJupyterPython notebook available as a sample file on the Data Science Virtual Machine. Note the language indicator for the notebook in the upper right corner of the browser window, showing that the notebook is attached to the Python 2 runtime. The circle to the right of "Python 2" indicates the current state of the kernel. A filled circle indicates that the kernel is in use and a program is executing. A hollow circle indicates the kernel is idle. Also take note that the main body of the notebook contains text as well as code and a graphed plot.

Creating a Jupyter Notebook in Azure Notebooks

There are several options to run a Jupyter notebook. However, the fastest way to get started is by using the Azure Notebook service, which is in preview mode at the time of this writing. Browse over to notebooks.azure.com and sign in with your Microsoft ID credentials. If prompted, grant the application the permissions it asks for. For first time users, the site will prompt you for a public user ID. This will create a URL to host your profile and to share notebooks. If you do not wish to set this up at this time, click "No Thanks." Otherwise, enter a value and click Save.

The screen will now show your profile page. The Azure Notebook service stores Jupyter Notebooks in Libraries. In order to create a notebook, first you must create a library. Under Libraries, there's a button to add a library. Click on it to create a new library. In the dialog box that follows, enter a name for the Library and an ID for the URL to share it. If you wish to make this library public, check the box next to "Public Library." Checking or unchecking the box next to "Create a README.md" will automatically insert a README.md file for documentation purposes. Click Create to create a new library.

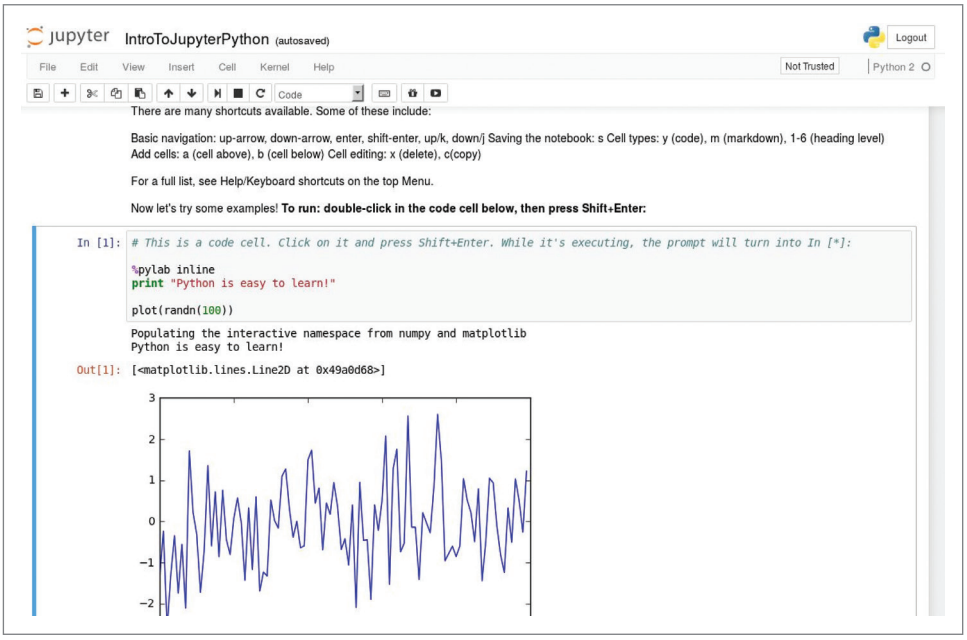


Figure 1 **Tutorial Notebook Introducing the Core Features of a Jupyter Notebook**

Now, your profile page will have one library listed. Click on it to bring up the contents of the library. Right now, the only item is the README.md file. Click on the New button to add a new item. In the ensuing dialog, enter a name for the new item and choose Python 3.6 Notebook from the drop down list next to Item Type and click New.

Once the item is created, it will appear in the library with a .IPYNB file extension. Click on it to launch an instance of the Jupyter server in

Azure. Note that a new browser tab or window will open and that the interface looks more like the screen in **Figure 1**. Click inside the text box and write the following code:

```
print("Hello World!")
```

Choose Run Cells from the Cell menu at the top of the page. The screen should look like **Figure 2**.

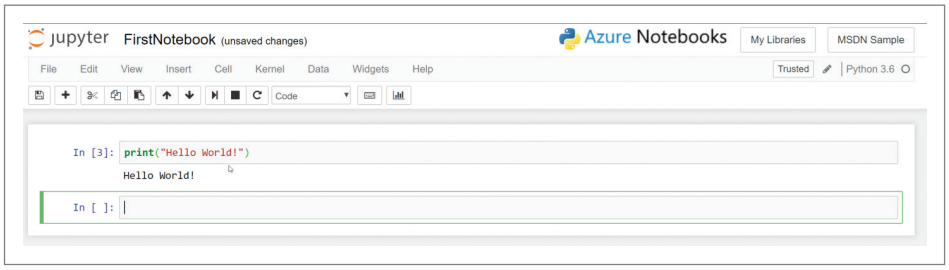


Figure 2 **Hello World! in a Jupyter Notebook**

Click inside the blank cell that Jupyter added and choose Cell > Cell Type > Markdown from the menu bar. Then add the following text.

```
# This is markdown text #
Markdown is a versatile and lightweight markup language.
```

Click the save icon and close the browser tab. Back in the Library window, click on the notebook file again. The page will reload and the markdown formatting will take effect.

Next, add another code cell, by clicking in the markdown cell and choosing Insert Cell Below from the Insert menu. Previously, I stated that only Python code could be executed in Python notebook. That is not entirely true, as you can use the "!" command to issue shell commands. Enter the following command into this new cell.

```
! ls -l
```

Choose Run Cells from the Run menu or click the icon with the Play/Pause symbol on it. The command returns a listing of the contents of the directory, which contains the notebook file and the README.md file. Once again, Jupyter added a blank cell after the response. Type the following code into the blank cell:

```
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt
x = np.random.rand(100)
y = np.random.rand(100)
plt.scatter(x, y)
plt.show()
```

Run the cell and after a moment a scatter plot will appear in the results. For those familiar with Python, the first line of code may look unfamiliar, as it is part of the IPython kernel which executes Python code in a Jupyter notebook. The command %matplotlib inline instructs the IPython runtime to display graphs generated by matplotlib in-line with the results. This type of command, known as a "magic" command, starts with "%". A full exploration of magic commands warrants its own article. For further information on magic commands, refer to the IPython documentation at bit.ly/2CfiMvh.

For those not familiar with Python, the previous code segment imports two libraries, NumPy and Matplotlib. NumPy is a Python package for scientific computing (numpy.org) and Matplotlib is a popular 2D graph plotting library for Python (matplotlib.org). The code then generates two arrays of 100 random numbers and plots the results as a scatter plot. The final line of code displays the graph. As the numbers are randomly generated, the graph will change slightly each time the cell is executed.

Notebooks in ML Workbench

So far, I have demonstrated running Jupyter networks as part of the Azure Notebook service in the cloud. However, Jupyter notebooks can be run locally, as well. In fact, Jupyter notebooks are integrated into the Azure Machine Learning Workbench product. In my previous article, I demonstrated the sample Iris Classification project. While the article did not mention it, there is a notebook included with the project that details all the steps needed to create a model, as well as a 3D plot of the iris dataset.

To view it, open the Iris Classifier sample project from last month's column, "Creating Models in Azure ML Workbench" (msdn.com/magazine/mt814992). If you did not create the project, follow the directions in the article to create a project from the template. Inside the project, shown in **Figure 3**, click on the third icon (1) from the top of the vertical toolbar on the left-hand side of the window, then click on iris (2) in the file list.

The notebook file loads, but the notebook server is not running—the results shown are cached from a previous execution. To make the notebook interactive, click on the Start Notebook Server (3) button to activate the local notebook server.

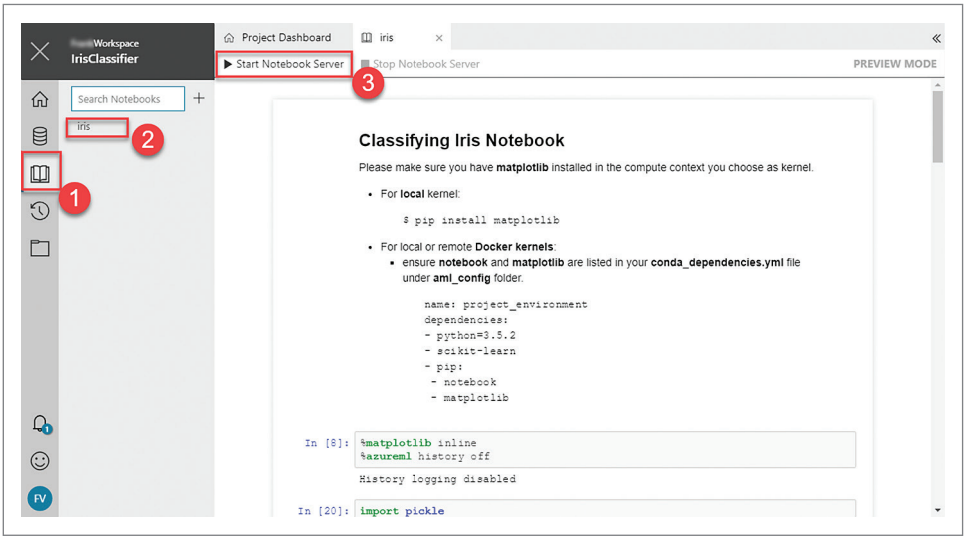


Figure 3 **Viewing Notebooks in an Azure Machine Learning Workbench Project**

```
In [17]: iris.corr()
```

Out[17]:

	Sepal Length	Sepal Width	Petal Length	Petal Width
Sepal Length	1.000000	-0.109369	0.871754	0.817954
Sepal Width	-0.109369	1.000000	-0.420516	-0.356544
Petal Length	0.871754	-0.420516	1.000000	0.962757
Petal Width	0.817954	-0.356544	0.962757	1.000000

Figure 4 **Correlation Matrix for the Iris Data Set**

Scroll down to the empty cell immediately following the 3D graph and enter the following code to view the first five records in the iris data set:

```
iris.head(5)
```

Choose Insert Cell Below from the Insert menu and enter the following code into the empty cell to display the correlation matrix:

```
iris.corr()
```

The output should look like **Figure 4**.

A correlation matrix displays the correlation coefficient between various fields in a data set. A correlation coefficient measures the linear dependence between two variables, with values closer to 1 indicating a positive correlation and values closer to -1 indicating a negative correlation. Values closer to 0 indicate a lack of correlation between the two fields. For example, there's a strong correlation between Petal Width and Petal Length with a value of 0.962757. On the other hand, the correlation between Sepal Width and Sepal Length is much weaker with a value of -0.109369. Naturally, each field has a 1.0 correlation with itself.

Anacondas

Thus far, I've only used Jupyter notebooks as part of either a Microsoft cloud service or locally using Microsoft software. However, Jupyter is open source and can run independent of the Microsoft ecosystem. One popular toolset is Anaconda (anaconda.com/download), an open source distribution of the Python and R for Windows, Mac and Linux. Jupyter ships as part of this install. Running Jupyter locally initializes a Web server locally on port 8888. Note that, on my system, I can only create a Python 3 notebook as that is the only kernel I have installed on my PC.

Data Science Virtual Machines

Running a Jupyter notebook server locally is ideal for scenarios where Internet access isn't reliable or guaranteed. For more compute-intensive tasks, it may be wiser to create a virtual machine and run Jupyter on more powerful hardware. To make this task easier, Azure offers the Data Science Virtual Machine image for both Windows and Linux, with the most popular data science tools already installed.

Creating a VM from this image is fast and simple. From the Azure Portal, click on the New icon and search for Data Science Virtual Machine" There are several options available. However, I've found that the Ubuntu image is the most feature-packed. Choose the Data Science Virtual Machine for Linux (Ubuntu) image and create a virtual machine by following the steps in the wizard. Once the machine is up and running, configure the VM for remote desktop access. Refer to documentation on how to connect to a Linux VM at bit.ly/2qgHOZo.

When connected to the machine, double-click on the Jupyter icon on the desktop. A terminal window will open, followed by a browser window a moment later. When clicking on the New button to create a new notebook, you have quite a few more choices of environments and languages, as demonstrated in **Figure 5**.

Along with the various runtime environments, the Data Science Virtual Machine for Ubuntu includes numerous sample notebooks. These notebooks provide guidance on everything from the basics of Azure ML to more advanced topics like CNTK and TensorFlow.

Wrapping Up

Jupyter notebooks are an essential tool for data science work, but they tend to confuse many developers because the platform lacks the basic features needed to develop software. This is by design. Jupyter notebooks are not intended for that task.

What notebooks do is provide a collaborative mechanism where data scientists can explore data sets, experiment with different hypotheses and share observations with colleagues. Jupyter notebooks can run locally on a PC, Mac or Linux. Azure ML Workbench even includes a notebook server embedded into the product for easier experimentation with data. Notebooks can also be run in the cloud as part of a service, such as Azure Notebooks, or on a VM with more capable hardware. ■

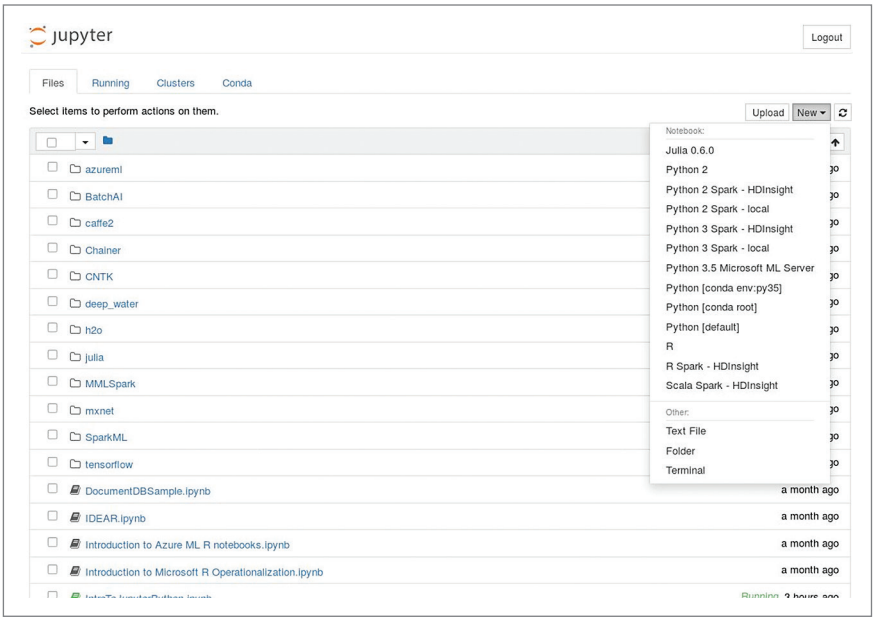


Figure 5 **Runtimes Available for the Data Science Virtual Machine for Ubuntu**

Frank La Vigne *leads the Data & Analytics practice at Wintellect and co-hosts the DataDriven podcast. He blogs regularly at FranksWorld.com and you can watch him on his YouTube channel, "Frank's World TV" (FranksWorld.TV).*

Thanks to the following technical experts for reviewing this article: *Andy Leonard*

File Format APIs

Open, Create, Convert, Print and Save files from your applications!

Try risk free - 30 day trial

Aspose.Total

Enable your applications to manipulate Word, Excel, PDF, PowerPoint, Outlook and more than 100 other file formats for all major platforms.

Aspose.Words

Create, edit, convert or print Word documents (DOC, DOCX, RTF etc.) in your .NET, Java and Android applications.

Aspose.Cells

Develop high performance .NET, Java and Android applications to Create, Edit or Convert Excel worksheets (XLS, XLSX, ODS etc).

Aspose.Pdf

Manipulate PDF file formats (PDF, PDF/A, XPS etc.) using our native APIs for .NET, Java and Android platforms.

Aspose.Slides

Create, edit or convert PowerPoint presentations (PPT, PPTX, ODP etc.) in your .NET, Java and Android applications.

Aspose.Email

Create, Edit or Convert Outlook Email file formats (MSG, PST, EML etc.) and popular network protocols.

Aspose.BarCode

Generate or recognize barcodes (Code128, PDF417, Postnet etc.) using our native APIs for .NET and Java.

Aspose.Imaging | Aspose.Tasks | Aspose.OCR | Aspose.Diagram | Aspose.Note | Aspose.HTML

ASPOSE File Format APIs

Download a Free Trial at **https://downloads.aspose.com**

Americas: +1 903 306 1676 EMEA: +44 141 628 8900 Oceania: +61 2 8006 6987
sales@asposeptyltd.com

C# 8.0 and Nullable Reference Types

Mark Michaelis

Nullable reference types—what? Aren't all reference types nullable?

I love C# and I find the careful language design fantastic. Nonetheless, as it currently stands, and even after 7 versions of C#, we still don't have a perfect language. By that I mean that while it's reasonable to expect there will likely always be new features to add to C#, there are also, unfortunately, some problems. And, by problems, I don't mean bugs but, rather, fundamental issues. Perhaps one of the biggest problem areas—and one that's been around since C# 1.0—surrounds the fact that reference types can be null and, in fact, reference types are null by default. Here are some of the reasons why nullable reference types are less than ideal:

- Invoking a member on a null value will issue a System.NullReferenceException exception, and every invocation that results in a System.NullReferenceException in production code is a bug. Unfortunately, however, with nullable reference types we "fall in" to doing the wrong thing rather than the right thing. The "fall in" action is to invoke a reference type without checking for null.
- There's an inconsistency between reference types and value types (following the introduction of Nullable<T>) in that value types are nullable when decorated with "?" (for example, int? number); otherwise, they default to non-nullable. In contrast, reference types are nullable by default. This is "normal" to those of us who have been programming in C# for a long time, but if we could do it all over, we'd want the default for reference types to be non-nullable and the addition of a "?" to be an explicit way to allow nulls.
- It's not possible to run static flow analysis to check all paths regarding whether a value will be null before dereferencing it, or not. Consider, for example, if there were unmanaged code invocations, multi-threading, or null assignment/replacement based on runtime conditions. (Not to mention whether analysis would include checking of all library APIs that are invoked.)
- There's no reasonable syntax to indicate that a reference type value of null is invalid for a particular declaration.
- There's no way to decorate parameters to not allow null.

As already said, in spite of all this, I love C# to the point that I just accept the behavior of null as an idiosyncrasy of C#. With C#

The C# Nullable Reference Types Preview is pre-release software. The features and behavior may change without notice.

This article discusses:

- The problem with nullable reference types
- Four improvements to C# 8.0 to fix nullable references
- C# 8.0 nullable reference type syntax
- Further enhancements in C# 8.0

Technologies discussed:

The Nullable Reference Type Design for C# 8.0

8.0, however, the C# language team is setting out with the intention of improving it. Specifically, they hope to do the following:

- **Provide syntax to expect null:** Enable the developer to explicitly identify when a reference type is expected to contain nulls—and, therefore, not flag occasions when it's explicitly assigned null.
- **Make default reference types expect non-nullable:** Change the default expectation of all reference types to be non-nullable, but do so with an opt-in compiler switch rather than suddenly overwhelm the developer with warnings for existing code.
- **Decrease the occurrence of NullReferenceExceptions:** Reduce the likelihood of NullReferenceException exceptions by improving the static flow analysis that flags potential occasions where a value hasn't been explicitly checked for null before invoking one of the value's members.
- **Enable suppression of static flow analysis warning:** Support some form of "trust me, I'm a programmer" declaration that allows the developer to override the static flow analysis of the complier and, therefore, suppress any warnings of a possible NullReferenceException.

For the remainder of the article, let's consider each of these goals and how C# 8.0 implements fundamental support for them within the C# language.

Provide the Syntax to Expect Null

To begin, there needs to be a syntax for distinguishing when a reference type should expect null and when it shouldn't. The obvious syntax for allowing null is using the ? as a nullable declaration—both for a value type and a reference type. By including support on reference types, the developer is given a way to opt-in for null with, for example:

```
string? text = null;
```

The addition of this syntax explains why the critical nullable improvement is summarized with the seemingly confusing name, "nullable reference types." It isn't because there's some new nullable reference data type, but rather that now there's explicit—opt-in—support for said data type.

> Just as with value types, reference types that allow null should be the exception—not the default.

Given the nullable reference types syntax, what's the non-nullable reference type syntax? While this:

```
string! text = "Inigo Montoya"
```

may seem a fine choice, it introduces the question of what is meant by simply:

```
string text = GetText();
```

Are we left with three declarations, namely: nullable reference types, non-nullable reference types and I-don't-know reference types? Ughh, No!!

Instead, what we really want is:

- Nullable reference types: string? text = null;
- Non-nullable reference types: string text = "Inigo Montoya"

This implies, of course, a breaking language change such that reference types with no modifier are non-nullable by default.

Make Default Reference Types Expect Non-Nullable

Switching standard reference declarations (no nullable modifier) to be non-nullable is perhaps the most difficult of all the require-

Further Enhancements in C# 8.0

There are three main additional areas of enhancement under consideration for C# 8.0:

Async Streams: Support for asynchronous streams enables await syntax to iterate over a collection of tasks (Task<bool>). For example, you can invoke

```
foreach await (var data in asyncStream)
```

and the thread will not block for any statements following the await, but will instead "continue with" them once the iterating completes. And the iterator will yield the next item upon request (the request is an invocation of Task<bool> MoveNextAsync on the enumerable stream's iterator) followed by a call to T Current { get; }.

Default Interface Implementations: With C# you can implement multiple interfaces such that the signatures of each interface are inherited. Furthermore, it's possible to provide a member implementation in a base class so that all derived classes have a default implementation of the member. Unfortunately, what's not possible is to implement multiple interfaces and also provide default implementations of the interface—that is, multiple inheritance. With the introduction of default interface implementations, we overcome the restriction. Assuming a reasonable default implementation is possible, with C# 8.0 you'll be able to include a default member implementation (properties and methods only) and all classes implementing the interface will have a default implementation. While multiple inheritance might be a side benefit, the real improvement this provides is the ability to extend interfaces with additional members without introducing a breaking API change. You could, for example, add a Count method to IEnumerator<T> (though implementing it would require iterating over all the items in the collection) without breaking all classes that implemented the interface. Note that this feature requires a corresponding framework release (something that hasn't been required since C# 2.0 and generics).

Extension everything: With LINQ came the introduction of extension methods. I recall having dinner with Anders Hejlsberg at the time and asking about other extension types, such as properties. Mr. Hejlsberg informed me that the team was only considering what was needed for implementing LINQ. Now, 10 years later, that assumption is being re-evaluated and they are considering the addition of extension methods for not only properties, but also for events, operators and even potentially constructors (the latter opens up some intriguing factory pattern implementations). The one important point to note—especially when it comes to properties, is that the extension methods are implemented in static classes and, therefore, there's no additional instance state for the extended type introduced. If you required such state, you'd need to store it in a collection indexed by the extended type instance, in order to retrieve the associated state.

ments to reduce nullable idiosyncrasy. The fact of the matter is that today, string text; results in a reference type called text that allows text to be null, expects text to be null, and, in fact, defaults text to be null in many cases, such as with a field or array. However, just as with value types, reference types that allow null should be the exception—not the default. It would be preferable if when we assigned null to text or failed to initialize text to something other than null, the compiler would flag any dereference of the text variable (the compiler already flags dereferencing a local variable before it's initialized).

Unfortunately, this means changing the language and issuing a warning when you assign null (string text = null, for example) or assign a nullable reference type (such as string? text = null; string moreText = text;). The first of these (string text = null) is a breaking change. (Issuing a warning for something that previously incurred no warning is a breaking change.) To avoid overwhelming developers with warnings as soon as they start using the C# 8.0 compiler, instead Nullability support will be turned off by default—thus no breaking change. To take advantage of it, therefore, you'll need to opt-in by enabling the feature. (Note, however, that in the preview available at the time of this writing, itl.tc/csnrtp, Nullability is on by default.)

> Support for nullable reference types should decrease the likelihood of throwing a NullReferenceException, though not eliminate it.

Of course, once the feature is enabled, the warnings will appear, presenting you with the choice. Choose explicitly whether the reference type is intended to allow nulls, or not. If it's not, then remove the null assignment, thus removing the warning. However, this potentially introduces a warning later on because the variable isn't assigned and you'll need to assign it a non-null value. Alternatively, if null is explicitly intended (representing "unknown" for example), then change the declaration type to be nullable, as in:

```
string? text = null;
```

Decrease the Occurrence of NullReferenceExceptions

Given a way to declare types as either nullable or non-nullable, it's now up to the compiler's static flow analysis to determine when the declaration is potentially violated. While either declaring a reference type as nullable or avoiding a null assignment to a non-nullable type will work, new warnings or errors may appear later on in the code. As already mentioned, non-nullable reference types will cause an error later on in the code if the local variable is never assigned (this was true for local variables before C# 8.0). In contrast, the static flow analysis will flag any dereference invocation of a nullable type for which it can't detect a prior check for null and/or any assignment of the nullable value to a value other than null. **Figure 1** shows some examples.

Figure 1 **Examples of Static Flow Analysis Results**

```
string text1 = null;
// Warning: Cannot convert null to non-nullable reference
string? text2 = null;
string text3 = text2;
// Warning: Possible null reference assignment
Console.WriteLine( text2.Length );
// Warning: Possible dereference of a null reference
if(text2 != null) { Console.WriteLine( text2.Length); }
// Allowed given check for null
```

Either way, the end result is a decrease in potential NullReferenceExceptions by using static flow analysis to verify a nullable intent.

As discussed earlier, the static flow analysis should flag when a non-nullable type will potentially be assigned null—either directly or when assigned a nullable type. Unfortunately, this isn't foolproof. For example, if a method declares that it returns a non-nullable reference type (perhaps a library that hasn't yet been updated with nullability modifiers) or one that mistakenly returns null (perhaps a warning was ignored), or a non-fatal exception occurs and an expected assignment doesn't execute, it's still possible that a non-nullable reference type could end up with a null value. That's unfortunate, but support for nullable reference types should decrease the likelihood of throwing a NullReferenceException, though not eliminate it. (This is analogous to the fallibility of the compiler's check when a variable is assigned.) Similarly, the static flow analysis won't always recognize that the code, in fact, does check for null before dereferencing a value. In fact, the flow analysis only checks the nullability within a method body of locals and parameters, and leverages method and operator signatures to determine validity. It doesn't, for example, delve into the body of a method called IsNullOrEmpty to run analysis on whether that method successfully checks for null such that no additional null check is required.

Enable Suppression of Static Flow Analysis Warning

Given the possible fallibility of the static flow analysis, what if your check for null (perhaps with a call such as object.ReferenceEquals(s, null) or string.IsNullOrEmpty()) is not recognized by the compiler? When the programmer knows better that a value isn't going to be null, they can dereference following the ! operator (for example, text!) as in:

```
string? text;
...
if(object.ReferenceEquals(text, null))
{
  var type = text!.GetType()
}
```

Without the exclamation point, the compiler will warn of a possible null invocation. Similarly, when assigning a nullable value to a non-nullable value you can decorate the assigned value with an exclamation point to inform the compiler that you, the programmer, know better:

```
string moreText = text!;
```

In this way, you can override the static flow analysis just like you can use an explicit cast. Of course, at runtime the appropriate verification will still occur.

Wrapping Up

The introduction of the nullability modifier for reference types doesn't introduce a new type. Reference types are still nullable and compiling string? results in IL that's still just System.String. The difference at the IL level is the decoration of nullable modified types with an attribute of:

```
System.Runtime.CompilerServices.NullableAttribute
```

In so doing, downstream compiles can continue to leverage the declared intent. Furthermore, assuming the attribute is available, earlier versions of C# can still reference C# 8.0-compiled libraries—albeit without any nullability improvements. Most important, this means that existing APIs (such as the .NET API) can be updated with nullable metadata without breaking the API. In addition, it means there's no support for overloading based on the nullability modifier.

> The transition of traditionally nullable declarations to non-nullable will initially introduce a significant number of warnings.

There's one unfortunate consequence to enhancing null handling in C# 8.0. The transition of traditionally nullable declarations to non-nullable will initially introduce a significant number of warnings. While this is unfortunate, I believe that a reasonable balance has been maintained between irritation and improving one's code:

- Warning you to remove a null assignment to a non-nullable type potentially eliminates a bug because a value is no longer null when it shouldn't be.
- Alternatively, adding a nullable modifier improves your code by being more explicit about your intent.
- Over time the impedance mismatch between nullable updated code and older code will dissolve, decreasing the NullReferenceException bugs that used to occur.
- The nullability feature is off by default on existing projects so you can delay dealing with it until a time of your choosing. In the end you have more robust code. For cases where you know better than the compiler, you can use the ! operator (declaring, "Trust me, I'm a programmer.") like a cast. ■

Mark Michaelis *is founder of IntelliTect, where he serves as its chief technical architect and trainer. For nearly two decades he has been a Microsoft MVP, and a Microsoft Regional Director since 2007. Michaelis serves on several Microsoft software design review teams, including C#, Microsoft Azure, SharePoint and Visual Studio ALM. He speaks at developer conferences and has written numerous books, including his most recent, "Essential C# 7.0 (6th Edition)" (itl.tc/EssentialCSharp). Contact him on Facebook at facebook.com/Mark.Michaelis, on his blog at IntelliTect.com/Mark, on Twitter: @markmichaelis or via e-mail at mark@IntelliTect.com.*

Thanks to the following Microsoft technical experts for reviewing this article: *Kevin Bost, Grant Ericson, Tom Faust, Mads Torgersen*

Event-Driven Architecture in the Cloud with Azure Event Grid

David Barkol

It's an exciting time to be a cloud architect. The pace of innovation has brought to the forefront a set of new challenges and technologies that are reshaping the way solutions are designed. Benefitting from this growth are the developers and architects who can choose from an abundance of services and options.

As developers go through the exercise of decomposing their architectures to take advantage of new services like Azure Functions, Logic Apps and others, familiar obstacles surface. In many cases, we find ourselves once again piecing together the "glue" that allows these services to work in concert. The recent launch of Azure Event Grid aims to solve this challenge by providing a first-class event routing service in the cloud that's fully managed, scalable and extremely flexible.

This article discusses:

- Design considerations for serverless and event-based architectures
- Benefits of using an event routing service for reliability and scale in the cloud
- Integrating multiple Azure services and events

Technologies discussed:

Microsoft Azure, Event Grid, Functions, Logic Apps, Visual Studio

Code download available at:

github.com/dbarkol/AzureEventGrid

In this article, I'll explore the flexibility of Azure Event Grid and show how it can be used to resolve familiar challenges in enterprise applications.

The Reversal of Dependencies

The notion of using events in a solution or application isn't new. In fact, event-driven programming has successfully utilized the concept for quite some time. Pub/Sub queues and GUIs are just a few of the examples that leverage this idea of reacting to events that happen in an organization, application or system.

> Azure Event Grid is a new, fully managed service that supports the routing of events by utilizing a publisher-subscriber model.

One of the core tenants of an event-driven architecture is to reverse the dependencies that existing services may have with each other. **Figure 1** shows an example of a set of processes that rely on each other to communicate and support a Human Resources (HR) department.

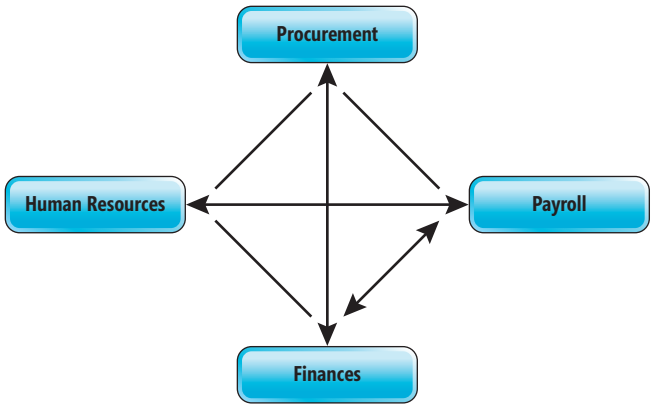


Figure 1 **Services That Contain Logic About Other Services**

For this design to work, each service must contain some basic logic about the others to communicate. These dependencies create challenges, not only in terms of scale, but also due to the fact that this logic is scattered across the architecture. Over time, as these types of solutions expand, they become difficult to maintain and increasingly brittle as more changes and dependencies are introduced.

As an alternative, the concept behind an event-driven design removes these dependencies by promoting the idea of an event as something that's a first-class citizen in the architecture. This important consideration allows many other systems the ability to leverage a centralized service without the burden of dependencies and logic dispersed throughout the application. **Figure 2** highlights the reversal of dependencies for the HR department solution by introducing this principal concept.

This key service is what the rest of the article is all about. I'll be exploring Azure Event Grid and how it can be used to support the next generation of solutions.

> At its core, Event Grid is an event routing service that manages the routing and delivery of events from numerous sources and subscribers.

Introducing Azure Event Grid

Azure Event Grid is a new, fully managed service that supports the routing of events by utilizing a publisher-subscriber model. At its core, Event Grid is an event routing service that manages the routing and delivery of events from numerous sources and subscribers. **Figure 3**, taken from the Event Grid overview documentation (bit.ly/2qhaj9q), illustrates several of the publishers and handlers that can be used with Event Grid today.

An event is created by a publisher such as a Blob Storage account, Event Hubs or even an Azure subscription. As events occur, they're published to an endpoint called a topic that the Event Grid service manages to digest all incoming messages. The list of services on Azure that integrate with Event Grid is growing, with many more on the horizon.

Event publishers aren't limited to services on Azure. In fact, a very common use case includes events that originate from custom applications or systems that can run from anywhere. This includes applications that are hosted on-premises, in a datacenter, or even on other clouds. These types of publishers are referred to as Custom Topics. If they can post an HTTP request to the Event Grid service, then they're candidates for sending events.

> Azure Event Grid is agnostic to any language or platform. While it integrates natively with Azure services, it can just as easily be leveraged by anything that supports the HTTP protocol, which makes it a very clever and innovative service.

Event handlers include several services on Azure, as well. These showcase some of the emerging serverless technologies on Azure, such as Functions and Logic Apps. In addition to Azure Automation, another type of event handler could be any HTTP callback, also referred to as a WebHook. Handlers are registered with Event Grid by creating an event subscription. If the event handler endpoint is publicly accessible and encrypted by Transport Layer Security, then messages can be pushed to it from Event Grid.

Unlike many other Azure services, there's no Event Grid namespace that needs to be provisioned or managed. Topics for native

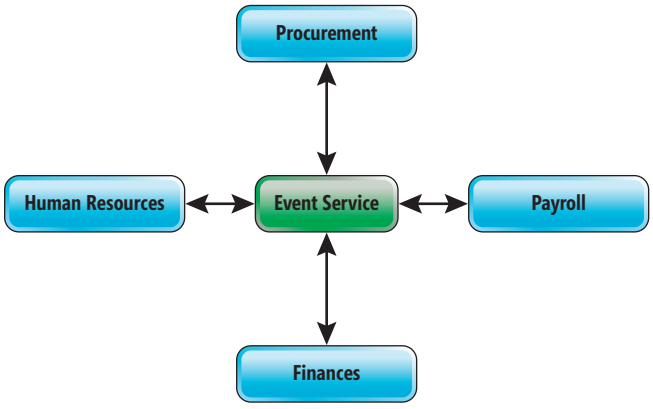


Figure 2 **A Centralized Service That Reverses Dependencies Between the Other Services**

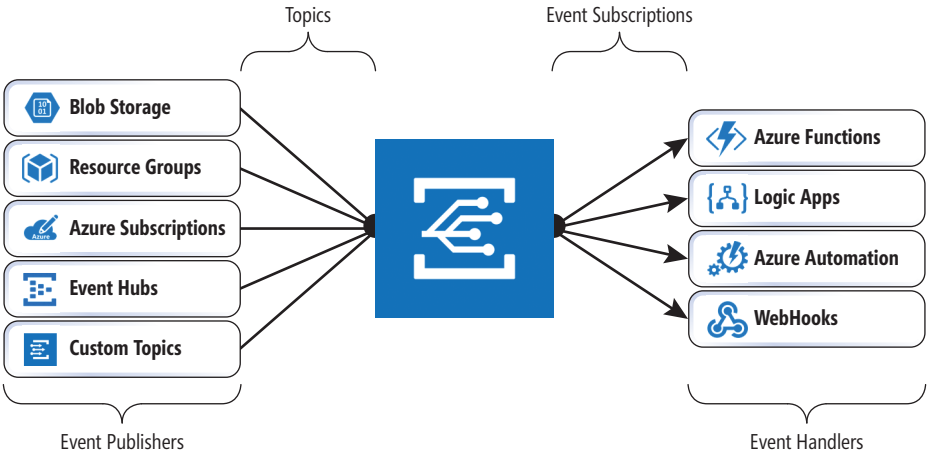


Figure 3 **Azure Event Grid Overview**

Azure resources are built in and completely transparent to users while custom topics are provisioned ad hoc and exist in a resource group. Event subscriptions are simply associated with a topic. This model simplifies management of topics as subscriptions and makes Event Grid highly multi-tenant, allowing for massive scale out.

Azure Event Grid is agnostic to any language or platform. While it integrates natively with Azure services, it can just as easily be leveraged by anything that supports the HTTP protocol, which makes it a very clever and innovative service.

Events or Commands

Before I dive into some code and build out a solution that highlights some of these features, lets distinguish between an event and a command. The distinction can sometimes be subtle, but it's important to comprehend when designing systems that rely on messages.

When a message is sent a specific action or response, it's most likely a command. For example, if an employee is promoted within an organization and a message is sent to instruct his new manager to fill out a form, then it carries with it a specific purpose or intent. Because the sender of the message has an expectation, and in some cases, might even expect a response, we can categorize this as a command.

If a message is published without any knowledge or expectations of how it will be handled, then it's deemed to be an event. Let's say that within an organization, the same employee has requested to change their mailing address. Because this action might be interesting to many systems in the organization but doesn't require the publisher to specifically be aware of any of them, it's a message that doesn't define any intent. In this case, the publisher is simply notifying any interested parties that an event has occurred. It's an event, and clearly an occurrence of something that's a viable option for a service like Event Grid.

I can spend a lot more time discussing these distinctions and in addition how to select the appropriate messaging service on Azure, however, that's out of the scope of this article. I encourage you to read this insightful post from Clemens Vasters on the topic: bit.ly/2CH3sbQ.

A Human Resources Scenario

The best way to get a deeper understanding of Event Grid is by writing code that leverages its capabilities. In this article, I'll look at a few events that originate from a fictitious HR application. I'll publish events to a custom topic and then subscribe to the events using several different handlers.

> When a message is sent a specific action or response, it's most likely a command.

To keep things simple, I'll be implementing two types of events from the HR system—when an employee is added to an organization and when an employee is removed. These events are close enough in nature that it will provide options that showcase how to filter and handle events in diverse ways. A visual representation of the solution is illustrated in **Figure 4**.

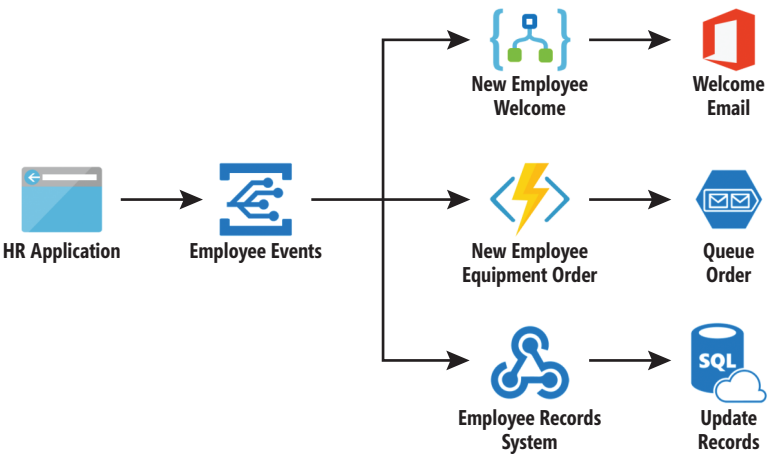


Figure 4 **A Sample Solution**

At a high-level, the solution consists of several key components that I will build in this article. Let's explore them here.

Employee Events will be an Event Grid Topic to which the HR application can send messages. This will include events for new and removed employees in the organization. Each message will contain information about the employee, her department and the type of event.

New Employee Welcome will be a Logic App that subscribes to messages for new employees in the organization. It will ultimately send a welcome email to the new employee.

DevExpress DXperience 17.2 | from $1,439.99

DevExpress

The complete range of DevExpress .NET controls and libraries for all major Microsoft platforms.

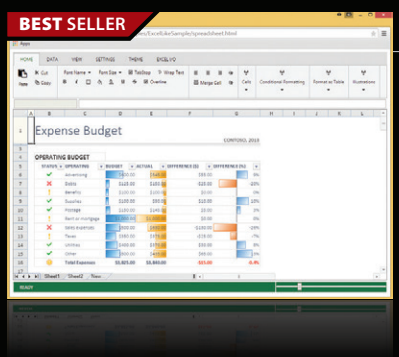
- WinForms - New TreeMap control, Chart series types and Unbound Data Source
- WPF - New Wizard control and Data Grid scrollbar annotations
- ASP.NET - New Vertical Grid control, additional Themes, Rich Editor Spell Checking and more
- Windows 10 Apps - New Hamburger Sub Menus, Splash Screen and Context Toolbar controls
- CodeRush - New debug visualizer expression map and code analysis diagnostics

Help & Manual Professional | from $586.04

ec software

Help and documentation for .NET and mobile applications.

- Powerful features in an easy, accessible and intuitive user interface
- As easy to use as a word processor, but with all the power of a true WYSIWYG XML editor
- Single source, multi-channel publishing with conditional and customized output features
- Output to responsive HTML, CHM, PDF, MS Word, ePUB, Kindle or print
- Styles and Templates give you full design control

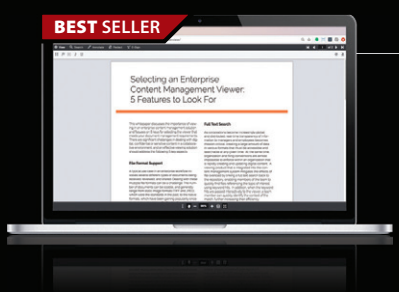


SpreadJS | from $1,476.52

GrapeCity

Deliver intuitive, efficient, multi-functional, pure JavaScript spreadsheets for Enterprise apps.

- Harness the power of a spreadsheet to display and manage data like Microsoft Excel
- Go beyond the grid with cards, trellis, calendar, Gantt, news feed, timeline and more
- Renders to the HTML canvas for a fast, interactive user experience across all browsers
- Modularized - so you only need to load the JavaScript that contains the features you need
- A Client-side component - works with Windows, Linux, MacOS, Android and iOS



PrizmDoc | from $576.24

accusoft

Web Services to add document & image processing functionality to your application.

- Collaboration tools for viewing, annotation, redaction, eSignature and document comparison
- Easy to embed into your Web site or application - choose Cloud-Hosted or Self-Hosted
- Compatible with any language/platform including ASP.NET, PHP, Ruby on Rails and Javascript
- Customizable HTML viewer UI makes it easy to match the style of your Web site or application
- Responsive HTML5 document viewer UI scales to fit desktop, laptop, tablet or phone.

We accept purchase orders.
Contact us to apply for a credit account.

USA
ComponentSource
650 Claremore Prof Way
Suite 100
Woodstock
GA 30188-5188
USA

Europe (UK)
ComponentSource
2 New Century Place
East Street
Reading, Berkshire
RG1 4ET
United Kingdom

Europe (Ireland)
ComponentSource
Unit 1E & 1F
Core B, Block 71
The Plaza, Park West
Dublin 12
Ireland

Asia Pacific
ComponentSource
7F Kojimachi Ichihara Bldg
1-1-8 Hirakawa-cho
Chiyoda-ku
Tokyo, 102-0093
Japan

Sales Hotline - US & Canada:
(888) 850-9911
www.componentsource.com

GSA Schedule Contract GS-35F-0188R

New Employee Equipment Order is an Azure Function that will subscribe to events for new employees in the Engineering department. It will then create a message in a queue for additional processing.

Employee Records is a custom Web site built on ASP.NET Core that will expose a Web API for receiving messages when employees leave the organization.

Creating a Custom Topic

To get started, I'll need to create a few basic resources in Azure. You can either launch the Azure Cloud Shell from the portal or use the command-line interface (CLI) locally. Details on how to use the Cloud Shell can be found at bit.ly/2CsFtQB. If you haven't used Cloud Shell before, I highly recommend it.

> Make note of the endpoint value—it will be used later when publishing events.

The first thing I'm going to do is create a resource group to manage and encapsulate the Azure resources:

```
az group create --name <resource-group-name> --location <location>
```

After the group is created, an Event Grid Topic is provisioned. This will provide an endpoint to publish custom events from the HR application. The name of the topic must be unique to the region, as it will be a publicly accessible service on Azure. The location must also be in a region that has the Event Grid service available. I usually go with the westus2 location or refer to the list of services provided in each Azure region (see bit.ly/2DU15In).

```
az eventgrid topic create --name <topic-name> \
  --location <location> \
  --resource-group <resource group name>
```

After executing the command to create the topic, details about the resource will be returned. The output will look similar to, but not exactly like, the code here:

```
{
  "endpoint": "https://<topic name>.westus2-1.eventgrid.azure.net/api/events",
  "id": "/subscriptions/xxxx-xxx-xx-xxx-xx/resourceGroups/
eventgridsolution-rg/providers/Microsoft.EventGrid/topics/<topic name>",
  "location": "westus2",
  "name": "<topic name>",
  "provisioningState": "Succeeded",
  "resourceGroup": "eventgridsolution-rg",
  "tags": null,
  "type": "Microsoft.EventGrid/topics"
}
```

Figure 5 **Employee Added Event**

```
[{
  "id": "30934",
  "eventType": "employeeAdded",
  "subject": "department/engineering",
  "eventTime": "2017-12-14T10:10:20+00:00",
  "data":{
    "employeeId": "14",
    "employeeName": "Nigel Tufnel",
    "employeeEmail": "nigel@contoso.com",
    "manager": "itmanager@contoso.com",
    "managerId": "4"
  }
}]
```

Make note of the endpoint value—it will be used later when publishing events. You'll also need one of the two access keys that were generated for authorization. To retrieve the keys, you can list the ones associated with the topic. You can and should cycle and regenerate these keys as a security measure—just like you would with other services on Azure.

```
az eventgrid topic key list --name <topic-name> --resource-group
<resource-group-name>
```

If your preference is to work within the Azure Portal, you can create and view all these options and settings there, as well.

Publishing an Event

Before sending the first event, you need to understand the event schema that's expected by the topic. Each event, regardless of if the publisher is an Azure resource or custom application, will adhere to the structure outlined in the following code (a helpful reference for the event schema, as well as some examples, can be found at bit.ly/2CG8oxI):

```
[
  {
    "topic": string,
    "subject": string,
    "id": string,
    "eventType": string,
    "eventTime": string,
    "data":{
      object-unique-to-each-publisher
    }
  }
]
```

The first thing to point out is that events are sent in an array. This is purposely done to provide the ability to send multiple events within a request. Events can be sent in batches, which reduces network chattiness while supporting scenarios where connectivity isn't always available.

The first event I want to publish is for when a new employee joins an organization. The payload for this event may resemble the contents of **Figure 5**.

Later in the article I'll use the same structure, with some differences in the values, as when an employee leaves the organization. The key properties in this event are as follows:

eventType is a value used to uniquely identify the published event type. This property can be used by handlers wishing to subscribe only to specific event types, rather than all types.

subject is a value, like eventType, that's available to provide additional context about the event, with the option of also providing an additional filter to subscribers. I'll leverage both eventType and subject soon when I create the subscriptions. Subject and eventType give context to the event.

data is a publisher-defined bucket that's simply an object that can contain one or more properties. Publishers assign relevant information about the event itself inside this property. For example, the Azure Blob storage event includes details about the created or deleted blob such as URL and content type.

To publish the event, I use Postman (or a similar tool) to simulate the message coming from the HR application to the endpoint address mentioned earlier. For authorization, I add an item in the header called aeg-sas-key—it's value is one of the access keys generated when the topic is created. The body of the request will contain the payload mentioned in **Figure 5**.

Figure 6 **Implementation for the New Employee Event Handler**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Threading.Tasks;
using Microsoft.Azure.WebJobs;
using Microsoft.Azure.WebJobs.Extensions.Http;
using Microsoft.Azure.WebJobs.Host;
using Newtonsoft.Json;

namespace NewEmployeeApp
{
  public static class NewEmployeeHandler
  {
    public class GridEvent<T> where T : class
    {
      public string Id { get; set; }
      public string EventType { get; set; }
      public string Subject { get; set; }
      public DateTime EventTime { get; set; }
      public T Data { get; set; }
      public string Topic { get; set; }
    }

      [FunctionName("newemployeehandler")]
      public static async Task<HttpResponseMessage> Run(
        [HttpTrigger(AuthorizationLevel.Function, "post", Route = null)]
        HttpRequestMessage req,
        TraceWriter log)
         {
          log.Info("New Employee Handler Triggered");

          // Retrieve the contents of the request and
          // deserialize it into a grid event object.
          var jsonContent = await req.Content.ReadAsStringAsync();
          var gridEvent =
            JsonConvert.DeserializeObject<List<GridEvent<Dictionary<string,
              string>>>>(jsonContent)
              ?.SingleOrDefault();

            // Check to see if the event is available and
            // return an error response if its missing.
            if (gridEvent == null)
            {
              return req.CreateErrorResponse(HttpStatusCode.BadRequest,
                $@"Missing event details");
            }

          // Check the header to identify the type of request
          // from Event Grid. A subscription validation request
          // must echo back the validation code.
          var gridEventType = req.Headers.GetValues("Aeg-Event-Type").
            FirstOrDefault();
          if (gridEventType == "SubscriptionValidation")
          {
            var code = gridEvent.Data["validationCode"];
            return req.CreateResponse(HttpStatusCode.OK,
              new { validationResponse = code });
          }
          else if (gridEventType == "Notification")
          {
            // Pseudo code: place message into a queue
            // for further processing.
            return req.CreateResponse(HttpStatusCode.OK);
          }
          else
          {
            return req.CreateErrorResponse(HttpStatusCode.BadRequest,
              $@"Unknown request type");
          }
        }
  }
}
```

Because there aren't any subscribers, there isn't anything to observe yet. The next step will be to see this in action by creating a few event handlers for Event Grid to push the events to.

Handling Events with an Azure Function

Now comes the fun part of subscribing to events. Our first handler will be an Azure Function. To learn the basics of creating a Function, see bit.ly/2A6pFgu. For this situation, I want to specifically subscribe to events for recently added employees. Additionally, and just as important, this handler must only be invoked for employees that belong to theengineering department.

Most examples walk through the creation of a Function using the Azure Portal—which is super-easy and quick. I'd like to show you how to do this locally, from Visual Studio. This will pave the way for more production-ready code. I'll also use a utility called ngrok (see ngrok.com) to support local debugging with Event Grid.

If you'd like to follow along, you'll need ngrok, as well as an up-to-date version of Visual Studio (I used version 15.5.2 for this article). Let's start by creating a new project and selecting Azure Functions from the Cloud templates. In the New Project dialog box, select the HTTP trigger option and keep the default values.

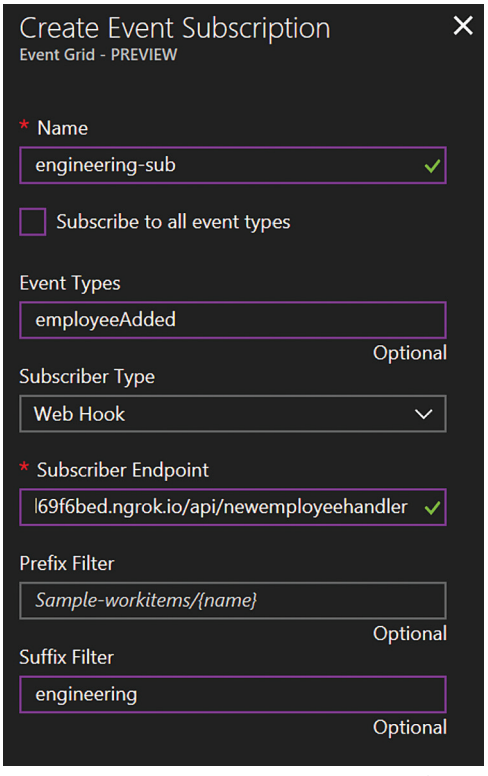


Figure 7 **Creating an Event Subscription from the Portal**

Update the code for the function to reflect what's represented in **Figure 6**. Feel free to rename the file to reflect the function name.

There is some important code to review here. At the very beginning is a class called GridEvent that's intended to reflect the payload and event schema from Event Grid. Ideally, I would place this class in a common library so it can be reused. For this example, it's used to deserialize the contents of the request into a strongly typed object.

Event Grid will send to its subscribers two types of requests—SubscriptionValidation and Notification—that you can identify by inspecting a value from the header. The validation request is important to ensure that all subscribers are added explicitly. All I must do here is echo back the validation code to acknowledge that I can receive messages:

```
var code = gridEvent.Data["validationCode"];
return req.CreateResponse(HttpStatusCode.OK,
  new { validationResponse = code });
```

Validation requests can also be identified by their event type: Microsoft.EventGrid.SubscriptionValidationEvent. If the event type is Notification, then I proceed with the implementation of the business logic. This defensive programming approach is highly recommended when exposing endpoints to other services.

Functions that are hosted in Azure, and are referenced with the azurewebsites.net domain, do not require the subscription validation logic. Instead, they're whitelisted by Event Grid along with several other services such as Logic Apps and callbacks from Azure Automation run books. Because I plan to test locally, I need to echo back the validation code for Event Grid to acknowledge the function as a valid endpoint.

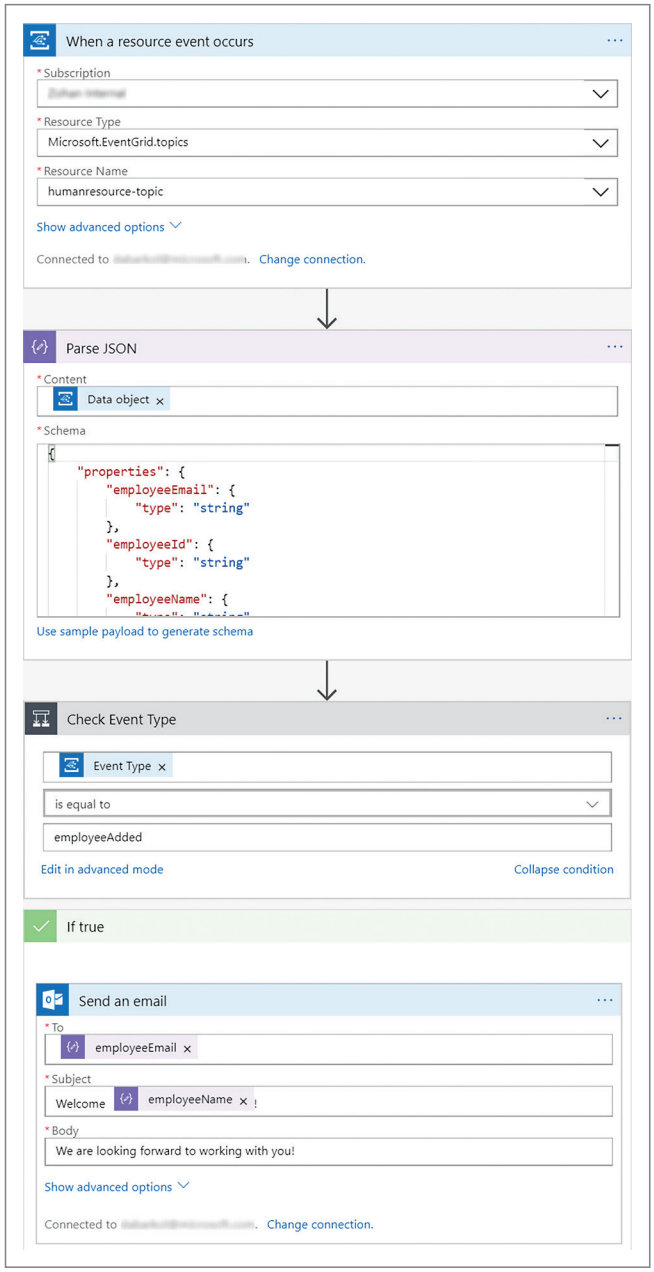


Figure 8 **A Logic App That Welcomes New Employees**

Eventually the Event Grid runtime SDK will handle much of this setup, from deserializing events and creating strongly typed Event Grid objects, to validating endpoints automatically. As of this writing, the updated runtime SDKs were not yet available.

Local Function Testing

Let's start the function from Visual Studio so that it runs locally on port 7071. Once it's running, open a command prompt and use ngrok to create a secure tunnel:

```
ngrok http -host-header=localhost 7071
```

I'll get back an HTTPS address from ngrok to use as the subscriber endpoint. The address should look something like https://d69f6bed.ngrok.io, but with a different subdomain each time the ngrok command is executed. Append the route of our function to the URL so that it resembles something like https://<generated-value>.ngrok.io/api/newemployeehandler. This will be the endpoint address for the event subscription.

With the function running and the secure tunnel in place, I can now create the event subscription from the CLI or Azure Cloud Shell:

```
az eventgrid event-subscription create --name <event-subscription-name> \
  --resource-group <resource group name> \
  --topic-name <topic name> \
  --subject-ends-with engineering \
  --included-event-type employeeAdded \
  --endpoint <function endpoint>
```

Optionally, I can add an event subscription from the portal by filling out the dialog, as shown in **Figure 7**.

I want to call out several important arguments in the creation of the event subscription.

subject-begins-with (Prefix Filter) is an optional argument that filters based on the prefix of the subject field in the events. It is a literal string match. Wildcard characters and regular expressions are not supported.

subject-ends-with (Suffix Filter) is an optional argument based on a suffix to filter events. Wildcard characters and regular expressions are not supported.

included-event-type (Event Types) is an optional list of event types to subscribe to. Each type is separated by a space.

Now I can return to the publishing event example earlier in the article to ensure that events are flowing through from Postman, to Event Grid, and ultimately to the local function. Feel free to change the values in the request to validate the filters are working as expected.

Handling Events: Logic App and WebHook

The next event subscription is a Logic App. Like the Azure Function example, it's only interested in the added employee event type. It won't leverage the prefix or suffix filters, because I want to send a message to employees from all departments. The completed version of the Logic App is shown in **Figure 8**.

The Logic App begins with an Event Grid trigger. Selecting Microsoft.EventGrid.topics as the resource type will allow me to pick from the custom topics in the subscription.

The Parse JSON action will be helpful in accessing the properties within the Data object. I'll use this sample payload to generate the schema:

Data Quality Made Easy.
Your Data, Your Way.

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

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

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

Microsoft® SQL Server® Integration Services

Microsoft Dynamics® CRM

Microsoft® .NET

Microsoft Azure

Microsoft® Excel

Start Your Free Trial

www.melissa.com/msft-pd

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

1-800-MELISSA

melissa™

Figure 9 **A Web API Controller That Receives Events**

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Newtonsoft.Json;

namespace EmployeeRecords.Controllers
{
  public class GridEvent<T> where T : class
  {
    public string Id { get; set; }
    public string Subject { get; set; }
    public string EventType { get; set; }
    public T Data { get; set; }
    public DateTime EventTime { get; set; }
  }

  [Produces("application/json")]
  [Route("api/EmployeeUpdates")]
  public class EmployeeUpdatesController : Controller
  {
    private bool EventTypeSubcriptionValidation
      => HttpContext.Request.Headers["aeg-event-type"].FirstOrDefault() ==
        "SubscriptionValidation";

    private bool EventTypeNotification
      => HttpContext.Request.Headers["aeg-event-type"].FirstOrDefault() ==
        "Notification";

    [HttpPost]
    public async Task<HttpResponseMessage> Post()
    {
      using (var reader = new StreamReader(Request.Body, Encoding.UTF8))
      {
        var jsonContent = await reader.ReadToEndAsync();
        var gridEvent =
          JsonConvert.DeserializeObject<List<GridEvent<Dictionary<string,
          string>>>>(jsonContent)
            .SingleOrDefault();

        if (gridEvent == null)
        {
          return new HttpResponseMessage { StatusCode = HttpStatusCode.BadRequest};
        }

        // Check the event type from Event Grid.
        if (EventTypeSubcriptionValidation)
        {
          // Retrieve the validation code and echo back.
          var validationCode = gridEvent.Data["validationCode"];
          var validationResponse =
            JsonConvert.SerializeObject(new { validationResponse =
            validationCode });

          return new HttpResponseMessage
          {
            StatusCode = HttpStatusCode.OK,
            Content = new StringContent(validationResponse)
          };
        }
        else if (EventTypeNotification)
        {
          // Pseudo code: Update records
          return new HttpResponseMessage { StatusCode = HttpStatusCode.OK };
        }
        else
        {
          return new HttpResponseMessage { StatusCode = HttpStatusCode.BadRequest };
        }
      }
    }
  }
}
```

```
{
  "id": "40000",
  "eventType": "employeeAdded",
  "subject": "department/finance",
  "eventTime": "2017-12-20T10:10:20+00:00",
  "data":{
    "employeeId": "24",
    "employeeName": "David St. Hubbins",
    "employeeEmail": "david@contoso.com",
    "manager": "finance@contoso.com",
    "managerId": "10"
  }
}
```

Next, the filter for the event type must be done with a condition action. This is a slight departure from how the event subscription was created with the Function because there isn't a way to select this option in the Event Grid trigger.

The last step sends an email to the employee. It uses the properties that were retrieved from the second step to populate the recipient address and subject fields of the email. To test the Logic App, click Run from the designer and send a message to the endpoint like before.

The last event subscription is a basic HTTP callback, or WebHook. I'll update an existing ASP.NET Core application with a Web API for incoming events. The code for the WebHook will be very similar to the Azure Function I wrote earlier. Some subtle differences include the way header values are retrieved to inspect the request type, as seen in **Figure 9**.

When creating the event subscription, the event type registered should be employeeRemoved. This change satisfies the requirement that the handler only wants to receive messages for an employee that's removed from the organization. Also notice that neither the prefix nor suffix filters are used because the subscriber wants to be notified for each occurrence, regardless of the department:

```
az eventgrid event-subscription create --name <event-subscription-name> \
  --resource-group <resource group name> \
  --topic-name <topic name> \
  --included-event-type employeeRemoved \
  --endpoint <function endpoint>
```

Last, remember that the endpoint for the event subscription must be secure. If you reference an App Service on Azure, you have to specify HTTPS in the address, or adding the subscription will fail.

Wrapping Up

Azure Event Grid is truly a game-changing service. In this article, I addressed a common application integration scenario. Event Grid was used as the enabling technology to connect the application to other services such as an Azure Function, a Logic App and even a custom WebHook that could reside anywhere. When complemented with serverless apps, Event Grid really shines, as the two together can take advantage of the tremendous scale and integration features that Azure supports. The code in this article can be found at github.com/dbarkol/AzureEventGrid. ■

David Barkol *is an Azure Specialist at Microsoft on the Global Black Belt Team. Contact him on Twitter: @dbarkol or through email at dabarkol@microsoft.com.*

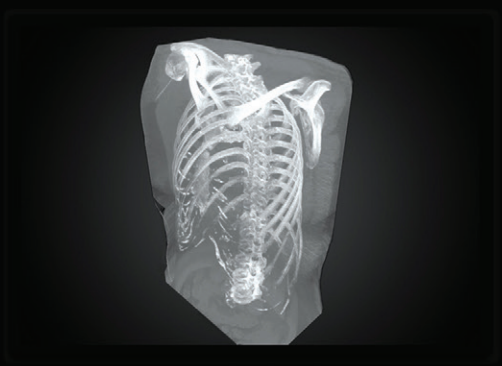
Thanks to the following Microsoft technical experts for reviewing this article: *Bahram Banisadr and Dan Rosanovsanova*

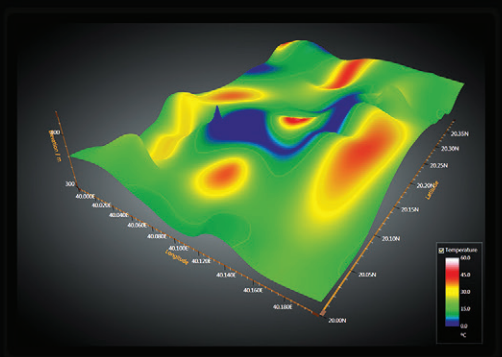
Arction

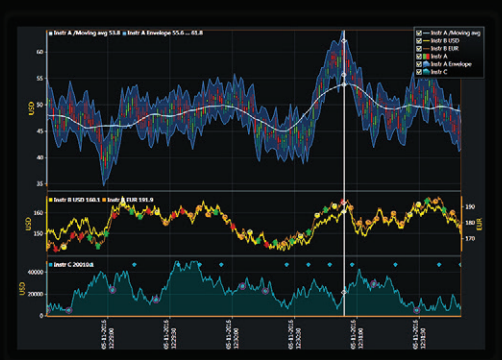
[WPF]
[Windows Forms]
[Free Gauges]
[Data Visualization]
[Volume Rendering]
[3D / 2D Charts] [Maps]

LightningChart®

The fastest and most advanced charting components







Create eye-catching and powerful charting applications for engineering, science and trading

- DirectX GPU-accelerated
- Optimized for real-time monitoring
- Supports gigantic datasets
- Full mouse-interaction
- Outstanding technical support
- Hundreds of code examples

NEW
- Now with Volume Rendering extension
- Flexible licensing options

Get free trial at
LightningChart.com/ms

Deep Neural Network Classifiers Using CNTK

James McCaffrey

The Microsoft Cognitive Toolkit (CNTK) library is a powerful set of functions that allows you to create machine learning (ML) prediction systems. I provided an introduction to version 2 in the July 2017 issue (msdn.com/magazine/mt784662). In this article, I explain how to use CNTK to make a deep neural network classifier. A good way to see where this article is headed is to take a look at the screenshot in **Figure 1**.

The CNTK library is written in C++ for performance reasons, but the most usual way to call into the library functions is to use the CNTK Python language API. I invoked the demo program by issuing the following command in an ordinary Windows 10 command shell:

```
> python seeds_dnn.py
```

This article discusses:

- Installing and using CNTK to create a deep neural network
- Understanding the data
- The deep neural network demo program
- Creating the network and the model
- Training the network and saving the trained model

Technologies discussed:

Microsoft Cognitive Toolkit, Anaconda Python distribution

Code download available at:

msdn.com/magazine/0218magcode

The goal of the demo program is to create a deep neural network that can predict the variety of a wheat seed. Behind the scenes, the demo program uses a set of training data that looks like this:

```
|properties 15.26 14.84 ... 5.22 |variety 1 0 0
|properties 14.88 14.57 ... 4.95 |variety 1 0 0
...
|properties 17.63 15.98 ... 6.06 |variety 0 1 0
...
|properties 10.59 12.41 ... 4.79 |variety 0 0 1
```

The training data has 150 items. Each line represents one of three varieties of wheat seed: "Kama," "Rosa" or "Canadian." The first seven numeric values on each line are the predictor values, often called attributes or features in machine learning terminology. The predictors are seed area, perimeter, compactness, length, width, asymmetry coefficient, and groove length. The item-to-predict (often called the class or the label) fills the last three columns and is encoded as 1 0 0 for Kama, 0 1 0 for Rosa, and 0 0 1 for Canadian.

The demo program also uses a test data set of 60 items, 20 of each seed variety. The test data has the same format as the training data.

The demo program creates a 7-(4-4-4)-3 deep neural network. The network is illustrated in **Figure 2**. There are seven input nodes (one for each predictor value), three hidden layers, each of which has four processing nodes, and three output nodes that correspond to the three possible encoded wheat seed varieties.

The demo program trains the network using 5000 batches of 10 items each, using the stochastic gradient descent (SGD) algorithm. After the prediction model has been trained, it's applied to the 60-item test data set. The model achieved 78.33 percent accuracy, meaning it correctly predicted 47 of the 60 test items.

```
C:\WINDOWS\system32\cmd.exe

C:\DeepSeeds>python seeds_dnn.py

Begin wheat seed classification demo

Using CNTK verson = 2.2

Creating a 7-(4-4-4)-3 tanh softmax NN for seed data
Selected CPU as the process wide default device.
Creating a cross entropy, SGD with LR=0.01, batch=10 Trainer

Starting training

batch      0: mean loss = 1.1185, mean accuracy = 40.00%
batch   1000: mean loss = 0.2955, mean accuracy = 90.00%
batch   2000: mean loss = 0.4483, mean accuracy = 80.00%
batch   3000: mean loss = 0.6653, mean accuracy = 70.00%
batch   4000: mean loss = 0.3678, mean accuracy = 90.00%

Training complete

Evaluating test data

Classification accuracy on the 60 test items = 75.00%

Predicting variety for (non-normalized) seed features:
[ 17.6  15.9   0.8   6.2   3.5   4.1   6.1]

Raw output values are:
1.5262
2.4201
-2.4804

Prediction probabilities are:
0.2888
0.7060
0.0053

End demo


C:\DeepSeeds>
```

Figure 1 **Wheat Seed Variety Prediction Demo**

The demo program concludes by making a prediction for an unknown wheat seed. The seven input values are (17.6, 15.9, 0.8, 6.2, 3.5, 4.1, 6.1). The computed raw output node values are (1.0530, 2.5276, -3.6578) and the associated output node probability values are (0.1859, 0.8124, 0.0017). Because the middle value is largest, the output maps to (0, 1, 0) which is variety Rosa.

This article assumes you have intermediate or better programming skills with a C-family language, and a basic familiarity with neural networks. But regardless of your background, you should be able to follow along without too much trouble. The complete source code for the seeds_dnn.py program is presented in this article. The code, and the associated training and test data files, are also available in the file download that accompanies this article.

Installing CNTK v2

Because CNTK v2 is relatively new, you may not be familiar with the installation process. Briefly, you first install a Python language distribution (I strongly recommend the Anaconda distribution) which contains the core Python language and required Python packages, and then you install CNTK as an additional Python package. In other words, CNTK is not a standalone install.

At the time of this writing, the current version of CNTK is v2.3. Because CNTK is under vigorous development, by the time you read this, there could well be a newer version. I used the Anaconda distribution version 4.1.1 (which contains Python version 3.5.2, NumPy version 1.11.1, and SciPy version 0.17.1). After installing Anaconda, I installed the CPU-only version of CNTK using the pip utility program. Installing CNTK can be a bit tricky if you're careless with versioning compatibility, but the CNTK documentation describes the installation process in detail.

Understanding the Data

Creating most machine learning systems starts with the time-consuming and often annoying process of setting up the training and test data files. The raw wheat seeds data set can be found at bit.ly/2idhoRK. The raw 210-item tab-delimited data looks like this:

```
14.11  14.1   0.8911  5.42  3.302    2.7  5      1
16.63  15.46  0.8747  6.053 3.465    2.04 5.877  1
```

I wrote a utility program to generate a file in a format that can be easily handled by CNTK. The resulting 210-item file looks like:

```
|properties 14.1100 14.1000 ... 5.0000 |variety 1 0 0
|properties 16.6300 15.4600 ... 5.8770 |variety 1 0 0
```

The utility program added a leading "|properties" tag to identify the location of the features, and a "|variety" tag to identify the location of the class to predict. The raw class values were 1-of-N encoded (sometimes called one-hot encoding), tabs were replaced by single blank space characters, and all predictor values were formatted to exactly four decimals.

In most situations you'll want to normalize numeric predictor values so they all have roughly the same range. I didn't normalize this data, in order to keep this article a bit simpler. Two common forms of normalization are z-score normalization and min-max

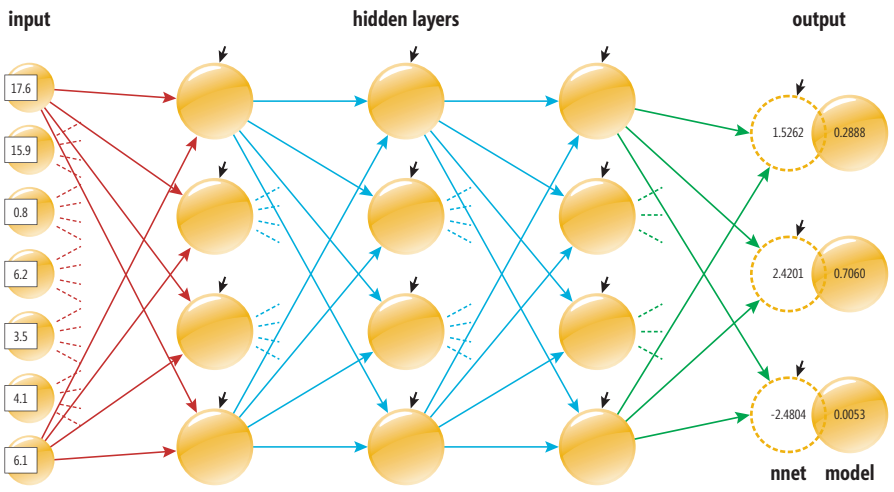


Figure 2 **Deep Neural Network Structure**

TEXT CONTROL

TX Text Control X15

Automate your reports and create beautiful documents in Windows Forms, WPF, ASP.NET and Cloud applications.

Text Control Reporting combines powerful reporting features with an easy-to-use, MS Word compatible word processor. Users create documents and templates using ordinary Microsoft Word skills.

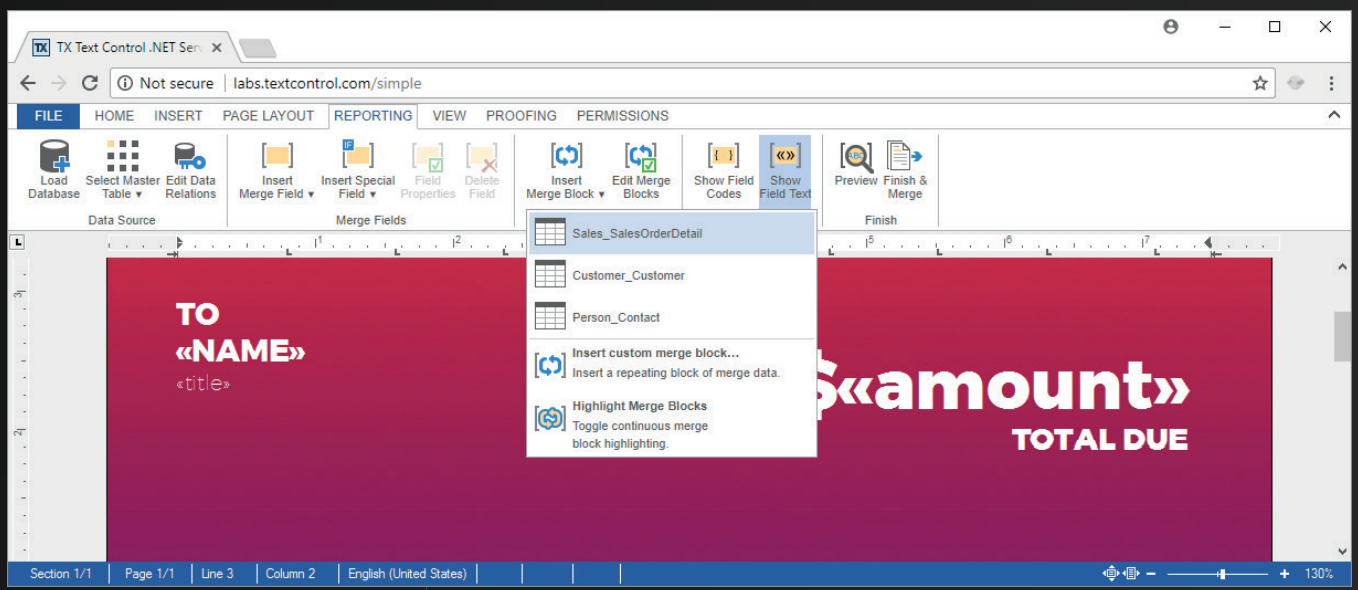
Download a free trial at
www.textcontrol.com

WE ARE CHANGING THE WAY YOU LOOK AT REPORTING

Visual Studio Partner

TX Text Control .NET Server for ASP.NET

Complete reporting and word processing for ASP.NET Web Forms and MVC



- Give your users a WYSIWYG, MS Word compatible, HTML5-based, cross-browser editor to create powerful reporting templates and documents anywhere.
- Text Control Reporting combines the power of a reporting tool and an easy-to-use WYSIWYG word processor - fully programmable and embeddable in your application.
- Replacing MS Office Automation is one of the most typical use cases. Automate, edit and create documents with Text Control UI and non-UI components.

WINDOWS FORMS | WPF | ASP.NET MVC | ASP.NET AJAX | CLOUD WEB API

TEXT CONTROL

normalization. In general, in non-demo scenarios you should normalize your predictor values.

Next, I wrote another utility program that took the 210-item data file in CNTK format, and then used the file to generate a 150-item training data file named seeds_train_data.txt (the first 50 of each variety) and a 60-item test file named seeds_test_data.txt (the last 20 of each variety).

Because there are seven predictor variables, it's not feasible to make a full graph of the data. But you can get a rough idea of the data's structure by the graph of partial data in **Figure 3**. I used just the seed perimeter and seed compactness predictor values of the 60-item test dataset.

The Deep Neural Network Demo Program

I used Notepad to write the demo program. I like Notepad but most of my colleagues prefer one of the many excellent Python editors that are available. The free Visual Studio Code editor with the Python language add-in is especially nice. The complete demo program source code, with a few minor edits to save space, is presented in **Figure 4**. Note that the backslash character is used by Python for line-continuation.

The demo begins by importing the required NumPy and CNTK packages, and assigning shortcut aliases of np and C to them. Function create_reader is a program-defined helper that can be used to read training data (if the is_training parameter is set to True) or test data (if is_training is set to False).

You can consider the create_reader function as boilerplate code for neural classification problems. The only things you'll need to change in most situations are the two string values of the field arguments in the calls to the StreamDef function, "properties" and "varieties" in the demo.

All the program control logic is contained in a single main function. All normal error checking code has been removed to keep the size of the demo small and to help keep the main ideas clear. Note that I indent two spaces rather than the more usual four spaces to save space.

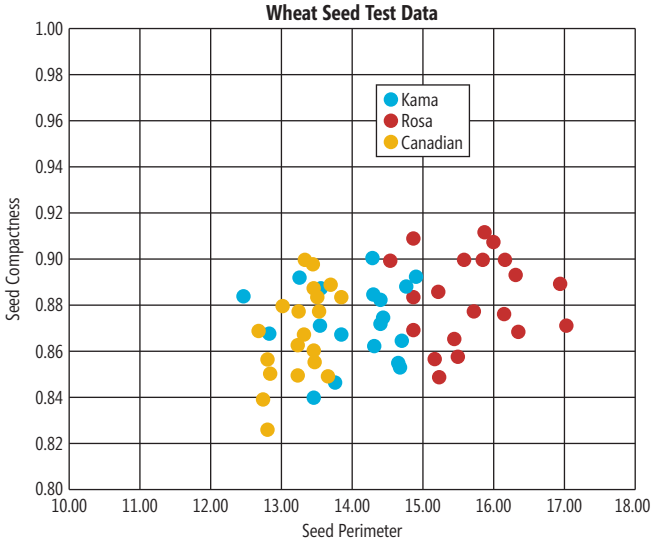


Figure 3 **Partial Graph of the Test Data**

Creating the Network and the Model

The main function begins by setting up the neural network architecture dimensions:

```
def main():
  print("Begin wheat seed classification demo")
  print("Using CNTK verson = " + str(C.__version__) )
  input_dim = 7
  hidden_dim = 4
  output_dim = 3
...
```

Because CNTK is under rapid development, it's a good idea to print out or comment the version being used. The demo has three hidden layers, all of which have four nodes. The number of hidden layers, and the number of nodes in each layer, must be determined by trial and error. You can have a different number of nodes in each layer if you wish. For example, hidden_dim = [10, 8, 10, 12] would correspond to a deep network with four hidden layers, with 10, 8, 10, and 12 nodes respectively.

Next, the location of the training and test data files is specified and the network input and output vectors are created:

```
train_file = ".\\Data\\seeds_train_data.txt"
test_file = ".\\Data\\seeds_test_data.txt"
# 1. create network and model
X = C.ops.input_variable(input_dim, np.float32)
Y = C.ops.input_variable(output_dim, np.float32)
```

Notice I put the training and test files in a separate Data subdirectory, which is a common practice because you often have many different data files during model creation. Using the np.float32 data type is much more common than the np.float64 type because the additional precision gained using 64 bits usually isn't worth the performance penalty you incur.

Next, the network is created:

```
print("Creating a 7-(4-4-4)-3 NN for seed data ")
with C.layers.default_options(init= \
  C.initializer.normal(scale=0.1, seed=2)):
  h1 = C.layers.Dense(hidden_dim,
    activation=C.ops.tanh, name='hidLayer1')(X)
  h2 = C.layers.Dense(hidden_dim, activation=C.ops.tanh,
    name='hidLayer2')(h1)
  h3 = C.layers.Dense(hidden_dim, activation=C.ops.tanh,
    name='hidLayer3')(h2)
  oLayer = C.layers.Dense(output_dim, activation=None,
    name='outLayer')(h3)
nnet = oLayer
model = C.softmax(nnet)
```

There's a lot going on here. The Python with statement is shortcut syntax to apply a set of common values to multiple layers of a network. Here, all weights are given a Gaussian (bell-shaped curve) random value with a standard deviation of 0.1 and a mean of 0. Setting a seed value ensures reproducibility. CNTK supports a large number of initialization algorithms, including "uniform," "glorot," "he" and "xavier." Deep neural networks are often surprisingly sensitive to the choice of initialization algorithm, so when training fails, one of the first things to try is an alternative initialization algorithm.

The three hidden layers are defined using the Dense function, so named because each node is fully connected to the nodes in the layers before and after. The syntax used can be confusing. Here, X acts as input to hidden layer h1. The h1 layer acts as input to hidden layer h2, and so on.

Notice that the output layer uses no activation function so the output nodes will have values that don't necessarily sum to 1. If you have experience with other neural network libraries, this requires some explanation. With many other neural libraries you'd use softmax activation on the output layer so that output value always sums

to 1 and can be interpreted as probabilities. Then, during training, you'd use cross-entropy error (also called log loss), which requires a set of values that sums to 1.

But, somewhat surprisingly, CNTK v2.3 doesn't have a basic cross-entropy error function for training. Instead, CNTK has a cross entropy with softmax function. This means that during training, output node values are converted on the fly to probabilities using softmax to compute an error term.

So, with CNTK, you train a deep network on raw output node values, but when making predictions, if you want prediction probabilities as is usually the case, you must apply the softmax function explicitly. The approach used by the demo is to train on the "nnet" object (no activation in the output layer), but create an additional "model" object, with softmax applied, for use when making predictions.

Now, it is, in fact, possible to use softmax activation on the output layer, and then use cross entropy with softmax during training. This approach results in softmax being applied twice, first to raw output values and then again to the normalized output node values. As it turns out, although this approach will work, for rather complex technical reasons, training isn't as efficient.

Chaining hidden layers is feasible up to a point. For very deep networks, CNTK supports a meta function named Sequential that provides a shortcut syntax for creating multi-layered networks. The CNTK library also has a Dropout function that can be used

Figure 4 **Complete Seed Classifier Demo Program**

```
# seeds_dnn.py
# classify wheat seed variety

import numpy as np
import cntk as C

def create_reader(path, is_training, input_dim, output_dim):
  strm_x = C.io.StreamDef(field='properties',
    shape=input_dim, is_sparse=False)
  strm_y = C.io.StreamDef(field='variety',
    shape=output_dim, is_sparse=False)
  streams = C.io.StreamDefs(x_src=strm_x,
    y_src=strm_y)
  deserial = C.io.CTFDeserializer(path, streams)
  sweeps = C.io.INFINITELY_REPEAT if is_training else 1
  mb_source = C.io.MinibatchSource(deserial,
    randomize=is_training, max_sweeps=sweeps)
  return mb_source

def main():
  print("\nBegin wheat seed classification demo  \n")
  print("Using CNTK verson = " + str(C.__version__) + "\n")

  input_dim = 7
  hidden_dim = 4
  output_dim = 3

  train_file = ".\\Data\\seeds_train_data.txt"
  test_file = ".\\Data\\seeds_test_data.txt"

  # 1. create network and model
  X = C.ops.input_variable(input_dim, np.float32)
  Y = C.ops.input_variable(output_dim, np.float32)

  print("Creating a 7-(4-4-4)-3 tanh softmax NN for seed data ")

  with C.layers.default_options(init= \
    C.initializer.normal(scale=0.1, seed=2)):
    h1 = C.layers.Dense(hidden_dim, activation=C.ops.tanh,
      name='hidLayer1')(X)
    h2 = C.layers.Dense(hidden_dim, activation=C.ops.tanh,
      name='hidLayer2')(h1)
    h3 = C.layers.Dense(hidden_dim, activation=C.ops.tanh,
      name='hidLayer3')(h2)
    oLayer = C.layers.Dense(output_dim, activation=None,
      name='outLayer')(h3)
  nnet = oLayer
  model = C.softmax(nnet)

  # 2. create learner and trainer
  print("Creating a cross entropy, SGD with LR=0.01, \
    batch=10 Trainer \n")
  tr_loss = C.cross_entropy_with_softmax(nnet, Y)
  tr_clas = C.classification_error(nnet, Y)

  learn_rate = 0.01
  learner = C.sgd(nnet.parameters, learn_rate)
  trainer = C.Trainer(nnet, (tr_loss, tr_clas), [learner])

  max_iter = 5000  # maximum training iterations
  batch_size = 10   # mini-batch size

  # 3. create data reader
  rdr = create_reader(train_file, True, input_dim,
    output_dim)
  my_input_map = {
    X : rdr.streams.x_src,
    Y : rdr.streams.y_src
  }

  # 4. train
  print("Starting training \n")
  for i in range(0, max_iter):
    curr_batch = rdr.next_minibatch(batch_size,
      input_map=my_input_map)
    trainer.train_minibatch(curr_batch)
    if i % 1000 == 0:
      mcee = trainer.previous_minibatch_loss_average
      pmea = trainer.previous_minibatch_evaluation_average
      macc = (1.0 - pmea) * 100
      print("batch %6d: mean loss = %0.4f, \
        mean accuracy = %0.2f%% " % (i,mcee, macc))

  print("\nTraining complete")

  # 5. evaluate model on the test data
  print("\nEvaluating test data \n")

  rdr = create_reader(test_file, False, input_dim, output_dim)
  my_input_map = {
    X : rdr.streams.x_src,
    Y : rdr.streams.y_src
  }
  numTest = 60
  allTest = rdr.next_minibatch(numTest, input_map=my_input_map)
  acc = (1.0 - trainer.test_minibatch(allTest)) * 100
  print("Classification accuracy on the \
    60 test items = %0.2f%%" % acc)

  # (could save model here)

  # 6. use trained model to make prediction
  np.set_printoptions(precision = 4)
  unknown = np.array([[17.6, 15.9, 0.8, 6.2, 3.5, 4.1, 6.1]],
    dtype=np.float32)
  print("\nPredicting variety for (non-normalized) seed features: ")
  print(unknown[0])

  raw_out = nnet.eval({X: unknown})
  print("\nRaw output values are: ")
  for i in range(len(raw_out[0])):
    print("%0.4f " % raw_out[0][i])

  pred_prob = model.eval({X: unknown})
  print("\nPrediction probabilities are: ")
  for i in range(len(pred_prob[0])):
    print("%0.4f " % pred_prob[0][i])

  print("\nEnd demo \n ")

# main()

if __name__ == "__main__":
  main()
```

to help prevent model overfitting. For example, to add dropout to the first hidden layer, you could modify the demo code like this:

```
h1 = C.layers.Dense(hidden_dim, activation=C.ops.tanh,
  name='hidLayer1')(X)
d1 = C.layers.Dropout(0.50, name='drop1')(h1)
h2 = C.layers.Dense(hidden_dim, activation=C.ops.tanh,
  name='hidLayer2')(d1)
h3 = C.layers.Dense(hidden_dim, activation=C.ops.tanh,
  name='hidLayer3')(h2)
oLayer = C.layers.Dense(output_dim, activation=None,
  name='outLayer')(h3)
```

Many of my colleagues prefer to always use Sequential, even for deep neural networks that only have a few hidden layers. I prefer manual chaining, but this is just a matter of style.

Training the Network

After creating a neural network and model, the demo program creates a Learner object and a Trainer object:

```
print("Creating a Trainer \n")
tr_loss = C.cross_entropy_with_softmax(nnet, Y)
tr_clas = C.classification_error(nnet, Y)
learn_rate = 0.01
learner = C.sgd(nnet.parameters, learn_rate)
trainer = C.Trainer(nnet, (tr_loss, tr_clas), [learner])
```

You can think of a Learner as an algorithm and a Trainer as an object that uses the Learner algorithm. The tr_loss ("training loss") object defines how to measure error between network-computed output values and known correct output values in the training data. For classification, cross entropy is almost always used, but CNTK supports several alternatives. The "with_softmax" part of the function name indicates that the function expects raw output node values rather than values normalized with softmax. This is why the output layer doesn't use an activation function.

The tr_clas ("training classification error") object defines how the number of correct and incorrect predictions are calculated during training. CNTK defines a classification error (percentage of incorrect predictions) library function rather than a classification accuracy function used by some other libraries. So, there are two forms of error being calculated during training. The tr_loss error is used to adjust the weights and biases. The tr_clas is used to monitor prediction accuracy.

The Learner object uses the SGD algorithm with a constant learning rate set to 0.01. SGD is the simplest training algorithm but it's rarely the best-performing one. CNTK supports a large number of learner algorithms, some of which are very complex. As a rule of thumb, I recommend starting with SGD and only trying more exotic algorithms if training fails. The Adam algorithm (Adam isn't an acronym) is usually my second choice.

Notice the unusual syntax for creating a Trainer object. The two loss function objects are passed as a Python tuple, indicated by the parentheses, but the Learner object is passed as a Python list, indicated by square brackets. You can pass multiple Leaner objects to a Trainer, though the demo program passes just one.

The code that actually performs training is:

```
for i in range(0, max_iter):
  curr_batch = rdr.next_minibatch(batch_size,
    input_map=my_input_map)
  trainer.train_minibatch(curr_batch)
  if i % 1000 == 0:
    mcee = trainer.previous_minibatch_loss_average
    pmea = trainer.previous_minibatch_evaluation_average
    macc = (1.0 - pmea) * 100
    print("batch %6d: mean loss = %0.4f, \
      mean accuracy = %0.2f%% " % (i, mcee, macc))
```

It's important to monitor training progress because training often fails. Here, the average cross-entropy error on the just-used batch of 10 training items is displayed every 1,000 iterations. The demo displays the average classification accuracy (percentage of correct predictions on the current 10 items), which I think is a more natural metric than classification error (percentage of incorrect predictions).

Saving the Trained Model

Because there are only 150 training items, the demo neural network can be trained in just a few seconds. But in non-demo scenarios, training a very deep neural network can take hours, days or even longer. After training, you'll want to save your model so you won't have to retrain from scratch. Saving and loading a trained CNTK model is very easy. To save, you can add code like this to the demo program:

```
mdl = ".\\Models\\seed_dnn.model"
model.save(mdl, format=C.ModelFormat.CNTKv2)
```

The first argument passed to the save function is just a filename, possibly including a path. There's no required file extension, but using ".model" makes sense. The format parameter has the default value ModelFormat.CNTKv2, so it could've been omitted. An alternative is to use the new Open Neural Network Exchange format=ONNX.

Recall that the demo program created both an nnet object (with no softmax on the output) and a model object (with softmax). You'll normally want to save the softmax version of a trained model, but you can save the non-softmax object if you wish.

Once a model has been saved, you can load it into memory like so:

```
model = C.ops.functions.Function.Load(".\\Models\\seed_dnn.model")
```

And then the model can be used as if it had just been trained. Notice that there's a bit of asymmetry in the calls to save and load—save is a method on a Function object and load is a static method from the Function class.

Wrapping Up

Many classification problems can be handled using a simple feed-forward neural network (FNN) with a single hidden layer. In theory, given certain assumptions, an FNN can handle any problem a deep neural network can handle. However, in practice, sometimes a deep neural network is easier to train than an FNN. The mathematical basis for these ideas is called the universal approximation theorem (or sometimes the Cybenko Theorem).

If you're new to neural network classification, the number of decisions you have to make can seem intimidating. You must decide on the number of hidden layers, the number of nodes in each layer, an initialization scheme and activation function for each hidden layer, a training algorithm, and the training algorithm parameters such as learning rate and momentum term. However, with practice you'll quickly develop a set of rules of thumb for the types of problems with which you deal. ■

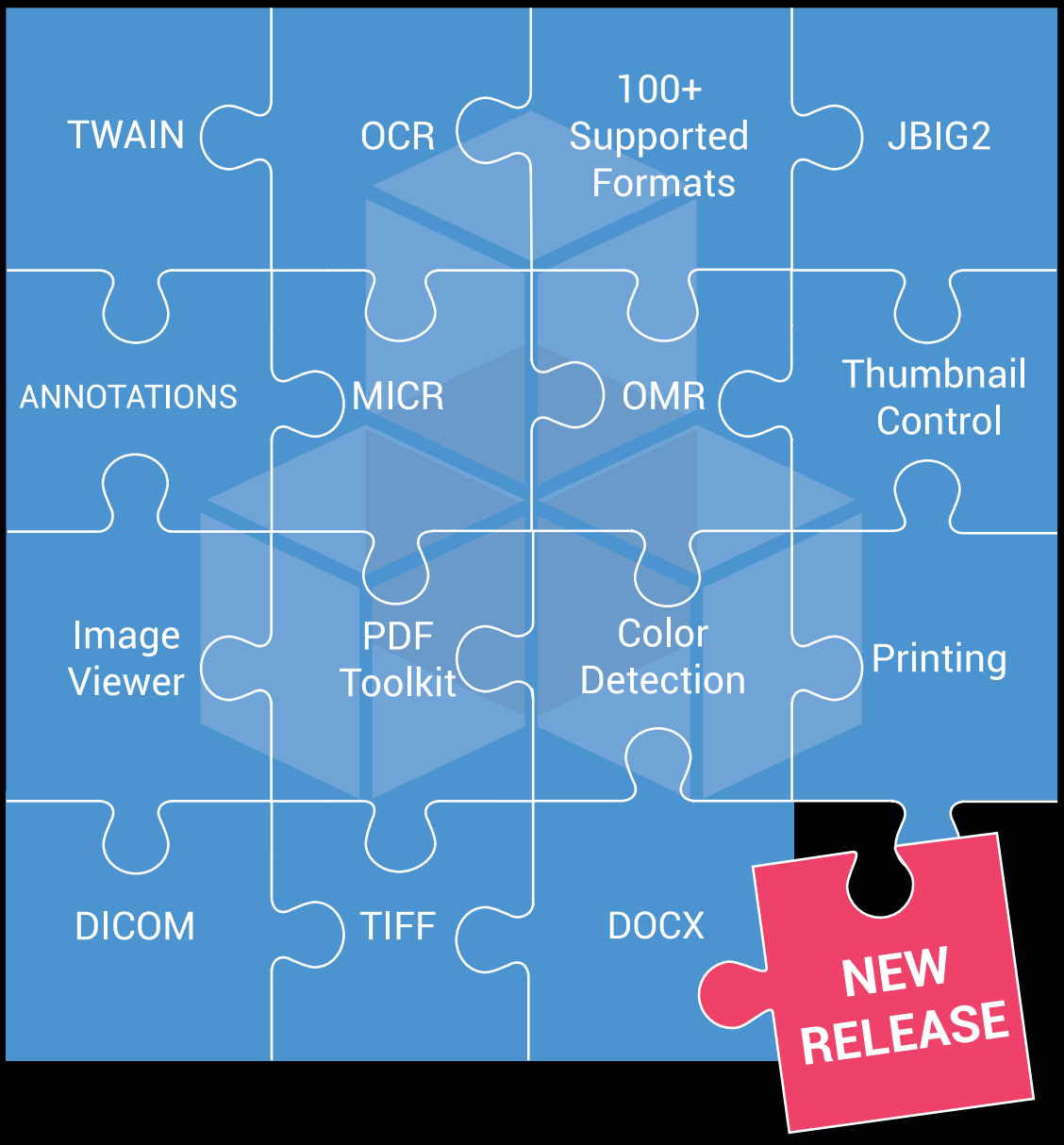
Dr. James McCaffrey *works for Microsoft Research in Redmond, Wash. He has worked on several Microsoft products, including Internet Explorer and Bing. Dr. McCaffrey can be reached at jamccaff@microsoft.com.*

Thanks to the following Microsoft technical experts who reviewed this article: *Chris Lee, Ricky Loynd, Kenneth Tran*

GdPicture.NET 14

100% ROYALTY FREE

Imaging SDK For **WinForms**, **WPF** And **Web** Development



Leverage your apps. with **GdPicture.NET** Imaging Toolkit

DOWNLOAD YOUR FREE TRIAL **www.gdpicture.com**

GdPicture.NET is an ORPALIS imaging technologies product

Respect the Past. Code the Future.

Visual Studio Live! (VSLive!™) Las Vegas, returns to the strip, March 11 – 16, 2018. During this intense week of developer training, you can sharpen your skills in everything from ASP.NET to Xamarin.

Plus, celebrate 25 years of coding innovation as we take a fun look back at technology and training since 1993. Experience the education, knowledge-share and networking at #VSLive25.

Visual Studio LIVE! 25 YEARS OF CODING INNOVATION

VSLive! 1998

VSLive! 2017

SUPPORTED BY

Microsoft | Visual Studio | msdn magazine | Visual Studio MAGAZINE

PRODUCED BY

1105 MEDIA
YOUR GROWTH. OUR BUSINESS.

DEVELOPMENT TOPICS INCLUDE:

- VS2017/.NET
- Angular/JavaScript
- ASP.NET Core
- Xamarin
- Azure / Cloud
- Hands-On Labs
- Software Practices
- ALM / DevOps
- SQL Server 2017
- UWP (Windows)

Who Should Attend and Why

We've been around since 1993. What's our secret? YOU! Since our first conference (VBITS/VSLive!/Visual Studio Live!), tens of thousands of developers, software architects, programmers, engineers, designers and more have trusted us year-in-and-year-out for unbiased and cutting-edge education on the Microsoft Platform.

Register to code with us today!

Register by February 16 and Save $300

Use Promo Code VSLFEB4

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

linkedin.com – Join the "Visual Studio Live" group!

vslive.com/lasvegasmsdn

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

AGENDA AT-A-GLANCE

BACK BY POPULAR DEMAND
Sunday Pre-Con Hands-On Labs
Choose From:

HOL01 Special 2-Day Hands-On Lab: Modern Security Architecture for ASP.NET Core
Sunday, March 11,
9:00am – 6:00pm (Part 1)*

Monday, March 12,
9:00am – 6:00pm (Part 2)*

Brock Allen
You will learn:

- The security architecture of ASP.NET Core
- About authenticating users with OpenID Connect
- How to protect Web APIs with OAuth2

*This 2-day Hands-On Lab is available with the six-day conference package or on its own. Details at **vslive.com/lasvegasmsdn**.

HOL02 From 0-60 in a Day with Xamarin and Xamarin.Forms
Introductory / Intermediate
Sunday, March 11,
9:00am – 6:00pm

Roy Cornelissen & Marcel de Vries
You will learn:

- How to build your first mobile apps on three platforms with the Xamarin framework
- How to maintain platform uniqueness while sharing a large chunk of your codebase
- How to think "mobile first" in your application architecture

HOL03 Busy Developer's HOL on Angular
Sunday, March 11,
9:00am – 6:00pm

Ted Neward
In this Hands-On Lab, we'll start from zero, with a little TypeScript, then start working with Angular 2: its core constructs and how it works with components, modules, and of course the ubiquitous model/view/controller approach.

ONLY $695 through February 16
Applies to HOL02 and HOL03 only.

ALM / DevOps | **Cloud Computing** | **Database and Analytics** | **Native Client**

START TIME	END TIME	
START TIME	**END TIME**	**Full Day Hands-On Labs: Sunday, March 11, 2018** *(Separate entry fee required)*
8:00 AM	9:00 AM	Pre-Conference Hands-On Lab Registration - Coffee and Morning Pastries
9:00 AM	6:00 PM	**HOL01** Full Day Hands-On Lab: Modern Security Architecture for ASP.NET Core (Part 1) - *Brock Allen*
4:00 PM	6:00 PM	Conference Registration Open
START TIME	**END TIME**	**Pre-Conference Workshops: Monday, March 12, 2018** *(Separate entry fee required)*
7:30 AM	9:00 AM	Pre-Conference Workshop Registration - Coffee and Morning Pastries
9:00 AM	6:00 PM	**HOL01** Full Day Hands-On Lab: Modern Security Architecture for ASP.NET Core (Part 2) - *Brock Allen*
7:00 PM	9:00 PM	Dine-A-Round
START TIME	**END TIME**	**Day 1: Tuesday, March 13, 2018**
7:00 AM	8:00 AM	Registration - Coffee and Morning Pastries
8:00 AM	9:15 AM	**T01** Go Serverless with Azure Functions - *Eric D. Boyd* / **T02** Getting Ready to Write Mobile Applications with Xamarin - *Kevin Ford*
9:30 AM	10:45 AM	**T05** Angular 101 - *Deborah Kurata* / **T06** Lessons Learned from Real World Xamarin.Forms Projects - *Nick Landry*
11:00 AM	12:00 PM	**KEYNOTE:** .NET Everywhere and for Everyone - *James Montemagno, Principal*
12:00 PM	1:00 PM	Lunch
1:00 PM	1:30 PM	Dessert Break - Visit Exhibitors
1:30 PM	2:45 PM	**T09** MVVM and ASP.NET Core Razor Pages - *Ben Hoelting* / **T10** Works On My Machine... Docker for Developers - *Chris Klug*
3:00 PM	4:15 PM	**T13** Angular Component Communication - *Deborah Kurata* / **T14** Customizing Your UI for Mobile Devices: Techniques to Create a Great User Experience - *Laurent Bugnion*
4:15 PM	5:30 PM	Welcome Reception
START TIME	**END TIME**	**Day 2: Wednesday, March 14, 2018**
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries
8:00 AM	9:15 AM	**W01** The Whirlwind Tour of Authentication and Authorization with ASP.NET Core - *Chris Klug* / **W02** Building Mixed Reality Experiences for HoloLens & Immersive Headsets in Unity - *Nick Landry*
9:30 AM	10:45 AM	**W05** TypeScript: The Future of Front End Web Development - *Ben Hoelting* / **W06** A Dozen Ways to Mess Up Your Transition From Windows Forms to XAML - *Billy Hollis*
11:00 AM	12:00 PM	**General Session:** The Act of Creation—How Dev Tooling Makes Successful
12:00 PM	1:00 PM	Birds-of-a-Feather Lunch
1:00 PM	1:30 PM	Dessert Break - Visit Exhibitors - Exhibitor Raffle @ 1:15pm (Must be present to win)
1:30 PM	1:50 PM	**W09** Fast Focus: 0-60 for Small Projects in Visual Studio Team Services - *Alex Mullans*
2:00 PM	2:20 PM	**W12** Fast Focus: HTTP/2: What You Need to Know - *Robert Boedigheimer*
2:30 PM	3:45 PM	**W15** Advanced Fiddler Techniques - *Robert Boedigheimer* / **W16** Building Cross-Platforms Business Apps with C# and CSLA .NET - *Rockford Lhotka*
4:00 PM	5:15 PM	**W19** Assembling the Web - A Tour of WebAssembly - *Jason Bock* / **W20** Radically Advanced XAML: Dashboards, Timelines, Animation, and More - *Billy Hollis*
7:00 PM	8:30 PM	VSLive! High Roller Evening Out
START TIME	**END TIME**	**Day 3: Thursday, March 15, 2018**
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries
8:00 AM	9:15 AM	**TH01** ASP.NET Core 2 For Mere Mortals - *Philip Japikse* / **TH02** Performance in 60 Seconds – SQL Tricks Everybody MUST Know - *Pinal Dave*
9:30 AM	10:45 AM	**TH05** Getting to the Core of ASP.NET Core Security - *Adam Tuliper* / **TH06** Secrets of SQL Server - Database Worst Practices - *Pinal Dave*
11:00 AM	12:00 PM	**Panel Discussion:** Security in Modern App Development - *Rockford Lhotka,*
12:00 PM	1:00 PM	Lunch
1:00 PM	2:15 PM	**TH09** Entity Framework Core 2 For Mere Mortals - *Philip Japikse* / **TH10** SQL Server 2017 - Intelligence Built-in - *Scott Klein*
2:30 PM	3:45 PM	**TH13** Busy Developer's Guide to Chrome Development - *Ben Hoelting* / **TH14** Introduction to Azure Machine Learning - *James McCaffrey*
4:00 PM	5:15 PM	**TH17** Securing Web Apps and APIs with IdentityServer - *Brian Noyes* / **TH18** Introduction to the CNTK v2 Machine Learning Library - *James McCaffrey*
START TIME	**END TIME**	**Post-Conference Workshops: Friday, March 16, 2018** *(Separate entry fee required)*
7:30 AM	8:00 AM	Post-Conference Workshop Registration - Coffee and Morning Pastries
8:00 AM	5:00 PM	**F01** Workshop: Creating Mixed Reality Experiences for HoloLens & Immersive Headsets with Unity - *Nick Landry & Adam Tuliper*

Speakers and sessions subject to change

Visual Studio LIVE! 25 YEARS OF CODING INNOVATION

March 11 – 16, 2018
Bally's Hotel & Casino

LasVegas

Software Practices	Visual Studio / .NET Framework	Web Client	Web Server

Full Day Hands-On Labs: Sunday, March 11, 2018 *(Separate entry fee required)*

Pre-Conference Hands-On Lab Registration - Coffee and Morning Pastries

HOL02 Full Day Hands-On Lab: From 0-60 in a Day with Xamarin and Xamarin.Forms - *Roy Cornelissen & Marcel de Vries*	**HOL03** Full Day Hands-On Lab: Busy Developer's HOL on Angular - *Ted Neward*

Conference Registration Open

Pre-Conference Workshops: Monday, March 12, 2018 *(Separate entry fee required)*

Pre-Conference Workshop Registration - Coffee and Morning Pastries

M02 Workshop: Developer Dive into SQL Server 2016 - *Leonard Lobel*	**M03** Workshop: Add Intelligence to Your Solutions with AI, Bots, and More - *Brian Randell*

Dine-A-Round

Day 1: Tuesday, March 13, 2018

Registration - Coffee and Morning Pastries

T03 Database Development with SQL Server Data Tools - *Leonard Lobel*	**T04** What's New in Visual Studio 2017 for C# Developers - *Kasey Uhlenhuth*
T07 Introduction to Azure Cosmos DB - *Leonard Lobel*	**T08** Using Visual Studio Mobile Center to Accelerate Mobile Development - *Kevin Ford*

Program Manager - Mobile Developer Tools, Microsoft

Lunch

Dessert Break - Visit Exhibitors

T11 DevOps for the SQL Server Database - *Brian Randell*	**T12** To Be Announced
T15 PowerShell for Developers - *Brian Randell*	**T16** To Be Announced

Welcome Reception

Day 2: Wednesday, March 14, 2018

Registration - Coffee and Morning Pastries

W03 Using Feature Toggles to Separate Releases from Deployments - *Marcel de Vries*	**W04** Lock the Doors, Secure the Valuables, and Set the Alarm - *Eric D. Boyd*
W07 Overcoming the Challenges of Mobile Development in the Enterprise - *Roy Cornelissen*	**W08** Computer, Make It So! - *Veronika Kolesnikova & Willy Ci*

Developers - *Kasey Uhlenhuth, Program Manager - .NET Managed Languages, Microsoft*

Birds-of-a-Feather Lunch

Dessert Break - Visit Exhibitors - Exhibitor Raffle @ 1:15pm (Must be present to win)

W10 Fast Focus: Cross Platform Device Testing with xUnit - *Oren Novotny*	**W11** Fast Focus: Understanding .NET Standard - *Jason Bock*
W13 Fast Focus: Serverless Computing: Azure Functions and Xamarin in 20 Minutes - *Laurent Bugnion*	**W14** Fast Focus: Why You Should Love SQL Server 2017 - *Scott Klein*
W17 Versioning NuGet and npm Packages - *Alex Mullans*	**W18** Getting to the Core of .NET Core - *Adam Tuliper*
W21 Encrypting the Web - *Robert Boedigheimer*	**W22** Porting MVVM Light to .NET Standard: Lessons Learned - *Laurent Bugnion*

VSLive! High Roller Evening Out

Day 3: Thursday, March 15, 2018

Registration - Coffee and Morning Pastries

TH03 Demystifying Microservice Architecture - *Miguel Castro*	**TH04** Cognitive Services in Xamarin Applications - *Veronika Kolesnikova*
TH07 Unit Testing Makes Me Faster: Convincing Your Boss, Your Co-Workers, and Yourself - *Jeremy Clark*	**TH08** Publish Your Angular App to Azure App Services - *Brian Noyes*

James McCaffrey, Damian Brady, Oren Novotny, Pinal Dave, Veronika Kolesnikova

Lunch

TH11 Writing Testable Code and Resolving Dependencies - DI Kills Two Birds with One Stone - *Miguel Castro*	**TH12** Signing Your Code the Easy Way - *Oren Novotny*
TH15 "Doing DevOps" as a Politically Powerless Developer - *Damian Brady*	**TH16** Analyzing Code in .NET - *Jason Bock*
TH19 I'll Get Back to You: Task, Await, and Asynchronous Methods - *Jeremy Clark*	**TH20** Multi-targeting the World: A Single Project to Rule Them All - *Oren Novotny*

Post-Conference Workshops: Friday, March 16, 2018 *(Separate entry fee required)*

Post-Conference Workshop Registration - Coffee and Morning Pastries

F02 Workshop: Distributed Cross-Platform Application Architecture - *Jason Bock & Rockford Lhotka*	**F03** Workshop: UX Design for Developers: Basics of Principles and Process - *Billy Hollis*

Bally's Hotel & Casino will play host to Visual Studio Live!, and is offering a special reduced room rate to conference attendees.

CONNECT WITH VISUAL STUDIO LIVE!

- twitter.com/vslive – @VSLive
- facebook.com – Search "VSLive"
- linkedin.com – Join the "Visual Studio Live" group!

REGISTER NOW

vslive.com/lasvegasmsdn

Writing Native Mobile Apps Using a Customizable Scripting Language

Vassili Kaplan

In the February 2016 issue of *MSDN Magazine*, I showed how to create a custom scripting language based on the Split-And-Merge algorithm for parsing mathematical expressions in C# (msdn.com/magazine/mt632273). I called my language Customizable Scripting in C#, or CSCS. Recently, I published an E-book that provided more details about creating a custom language (bit.ly/2yijCod). Creating your own scripting language might not initially seem to be particularly useful, even though there are some interesting applications of it (for example, game cheating). I also found some applications in Unity programming.

But then I discovered an even more interesting application for a customizable scripting language—writing cross-platform applications for mobile devices. It turns out that it's possible to use CSCS to write apps for Android and iOS (and Windows Phone can be easily added, as well). And the same code can be used for all platforms. I published an introduction on how to do that in the November-December 2017 issue of *CODE Magazine* (codemag.com/article/1711081).

In this article I'm going to take a deeper dive and show how to use CSCS to program for mobile devices. I'll also correct some inaccuracies in the *CODE Magazine* article. You'll see that anything that can be done on the native platform can be done in CSCS. I'm also going to show how you can add missing features to CSCS on the fly.

To run the code shown in this article, you'll need Visual Studio 2017 with Xamarin installed, either on Windows or on macOS. I personally use Visual Studio Community Edition 2017 on my MacBook. Note that a Mac is needed to deploy iOS apps to the Apple App Store.

This article discusses:

- A "Hello, World!" for mobile apps written in CSCS
- The general structure of Android and iOS projects
- Adding text-to-speech and voice recognition to CSCS
- A currency convertor example to highlight more CSCS features for cross-platform app development

Technologies discussed:

C#, Visual Studio 2017, Xamarin, Android, iOS

Code download available at:

github.com/vassilych/mobile

A "Hello, World!" for Mobile Apps

Take a look at **Figure 1**, which shows some basic CSCS code for text-to-speech and voice recognition. Let's examine the code line by line.

The AutoScale function lets you automatically adjust widget size based on the actual device screen size. For instance, with AutoScale, the width of a widget will be twice as large on a device with a width of 1280 pixels as on one with a width of 640 pixels. The actual signature of the AutoScale function is:

```
AutoScale(scale = 1.0);
```

If you don't use the default scale = 1.0 parameter, the specified scale parameter will be applied to the difference. For example, if

scale = 0.5 the difference in widget sizes when moving from 640 to 1280 pixels will be not twice but 1.5 times because the formula to calculate the new size is:

```
newSize = size + size * scale * (1280 / 640 - 1) = size * (1 + scale) = size * 1.5.
```

But if scale = 2, the widget will be 3 times larger according to the calculation. A special case of scale = 0 also satisfies the formula here: No scale adjustment will be performed—the widget will have exactly the same size regardless of the device size. This scale parameter can also be applied per widget—it can be specified as an optional parameter in the GetLocation function. I'll show how to do this in a bit.

Next, I define a voice variable. Note that CSCS is a Python-like scripting language—the type of the variable is inferred from the context, so the voice variable will be represented as a C# string behind the scenes.

Then I define a button. A widget definition in CSCS always takes two statements: The first specifies the location of the widget and the second is the actual definition of the widget. Behind the scenes, a UIButton widget is used for iOS and a Button is used for Android.

The general syntax for creating a location on the screen is:

```
GetLocation(ReferenceX, PlacementX, ReferenceY, PlacementY,
            AdjustmentX = 0, AdjustmentY = 0,
            ScaleOption = false, Scale = 0.0, Parent = null);
```

Here's the meaning of the arguments:

- **ReferenceX:** The name of another widget for horizontal placement. It can be the string "ROOT," meaning the parent widget or the main screen.
- **PlacementX:** A horizontal point relative to the widget indicated in ReferenceX. Possible values are listed at the end of these arguments.
- **ReferenceY:** The name of another widget for vertical placement. It can be the string "ROOT," meaning the parent widget or the main screen.
- **PlacementY:** A vertical point relative to the widget indicated in ReferenceY. Possible values are listed at the end these arguments.
- **AdjustmenX:** An additional horizontal movement of the widget in pixels. It can also be negative; the positive direction goes from left to right.
- **AdjustmenY:** An additional vertical movement of the widget in pixels. It can also be negative; the positive direction goes from top to bottom.
- **ScaleOption:** Indicates whether to apply a particular scaling option to the widget. If this option is false or not provided, the adjustment specified in the AutoScale function will be done. If the option is provided, the adjustment parameters and the size of the widget will be modified according to the Scale parameter.
- **Scale:** The measure to be used for adjusting the size of the widget. The functionality is the same as in the AutoScale function. As a matter of fact, the same code will be executed.
- **Parent:** The parent of the widget. If not specified, the widget will be added to the Main Layout on Android or to the Root View Controller on iOS (specifically to Window.RootViewController.View).

Possible values for the placement arguments are very similar to the Android RelativeLayout.LayoutParams class. They can be any of: "CENTER," "LEFT," "RIGHT," "TOP," "BOTTOM," "ALIGN_LEFT," "ALIGN_RIGHT," "ALIGN_TOP," "ALIGN_BOTTOM," "ALIGN_PARENT_TOP," "ALIGN_PARENT_BOTTOM."

These parameters are used for both horizontal and vertical placement on iOS and on Android. No XML or XAML knowledge is needed. And there's no iOS Storyboard to deal with.

Once the location is created, you place a widget in it. Here's the general syntax for doing so:

```
AddWidget(location, widgetName, initParameter, width, height);
```

AddButton is a particular case of such a function, where the initialization argument is the text shown on the button. Other examples of widget functions are AddLabel, AddView, AddCombobox and there are many more, as you'll see.

The AddAction function assigns an action to a button when the user clicks on it. It generally has the following syntax:

```
AddAction(widgetName, callbackFunction);
```

A callback function in CSCS always has two parameters, a sender and a context argument, a concept borrowed from C#.

A widget definition in CSCS always takes two statements: the first specifies the location of the widget and the second is the actual definition of the widget.

Inside of the talk_click function, first I call the ShowToast function, which calls a native Toast implementation on Android and a custom Toast-like implementation on iOS. The iOS implementation just constructs a small frame with a message and destroys it after a timeout.

Finally, I call to the voice recognition function:

```
VoiceRecognition("voice_recog", voice = "en-US");
```

The first parameter is the name of the callback function to call when the voice recognition is complete. The second parameter is the voice. It's optional and, by default, it's U.S. English. The voices are

Figure 1 **A "Hello, World!" in CSCS for Mobile Apps**

```
AutoScale();
voice = "en-US";

locButtonTalk = GetLocation("ROOT", "CENTER", "ROOT", "BOTTOM", 0, 0);
AddButton(locButtonTalk, "buttonTalk", "Click me", 200, 80);
AddAction(buttonTalk,  "talk_click");

function talk_click(sender, arg) {
  ShowToast("Please say your name...");
  VoiceRecognition("voice_recog", voice);
}

function voice_recog(errorStatus, recognized) {
  if (errorStatus != "") {
    AlertDialog("CSCS", "Error: " + errorStatus);
  } else {
    ShowToast("Word recognized: " + recognized);
    Speak("Hello, " + recognized, voice);
  }
}
```

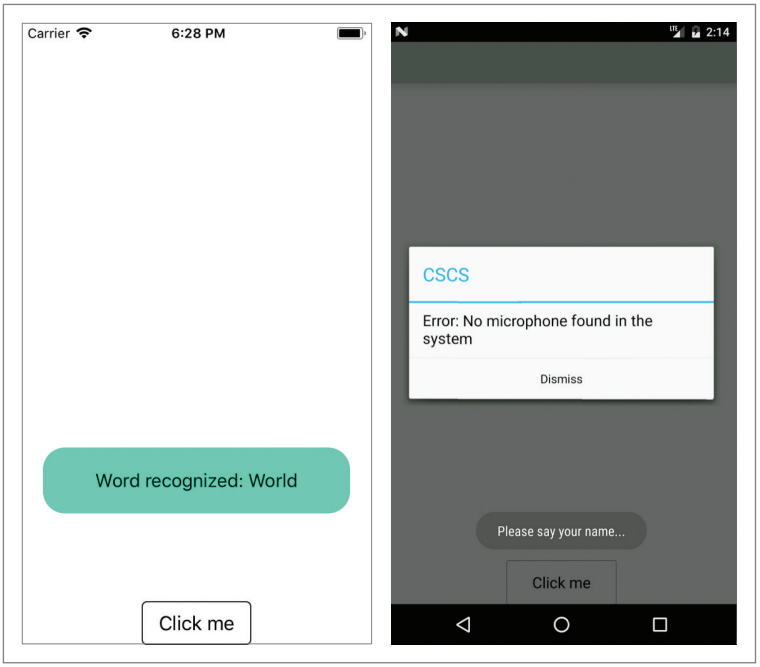


Figure 2 **An Example of Running the "Hello, World!" Script on iPhone (Left) and on Android (Right)**

specified as ISO 639-1 code for the language name and ISO 3166-1 alpha-2 for the country code (for example. "en-US" for English, U.S., "es-MX" for Spanish, Mexico, "pt-BR", for Portuguese, Brazil, and so on).

The signature of the voice recognition callback function is the following:

```
function voice_recog(errorStatus, recognized)
```

The errorStatus argument will be an empty string on success and a description of the error on failure. If the function is successful, the recognized word is passed as the second parameter. If not, an alert dialog will be shown to the user (implemented as a UIAlertController on iOS and as an AlertDialog.Builder on Android). If the voice recognition is successful, the text-to-speech function Speak will be called. It has the following signature:

```
Speak(wordToPronounce, voice = "en-US");
```

The results of running the script in **Figure 1** are shown in **Figure 2**. The figure on the left, representing an iPhone, shows successful voice recognition—when a pronounced word was recognized. The figure on the right, representing an Android, shows a failure, when there's no microphone installed on the system (a common case when using a simulator).

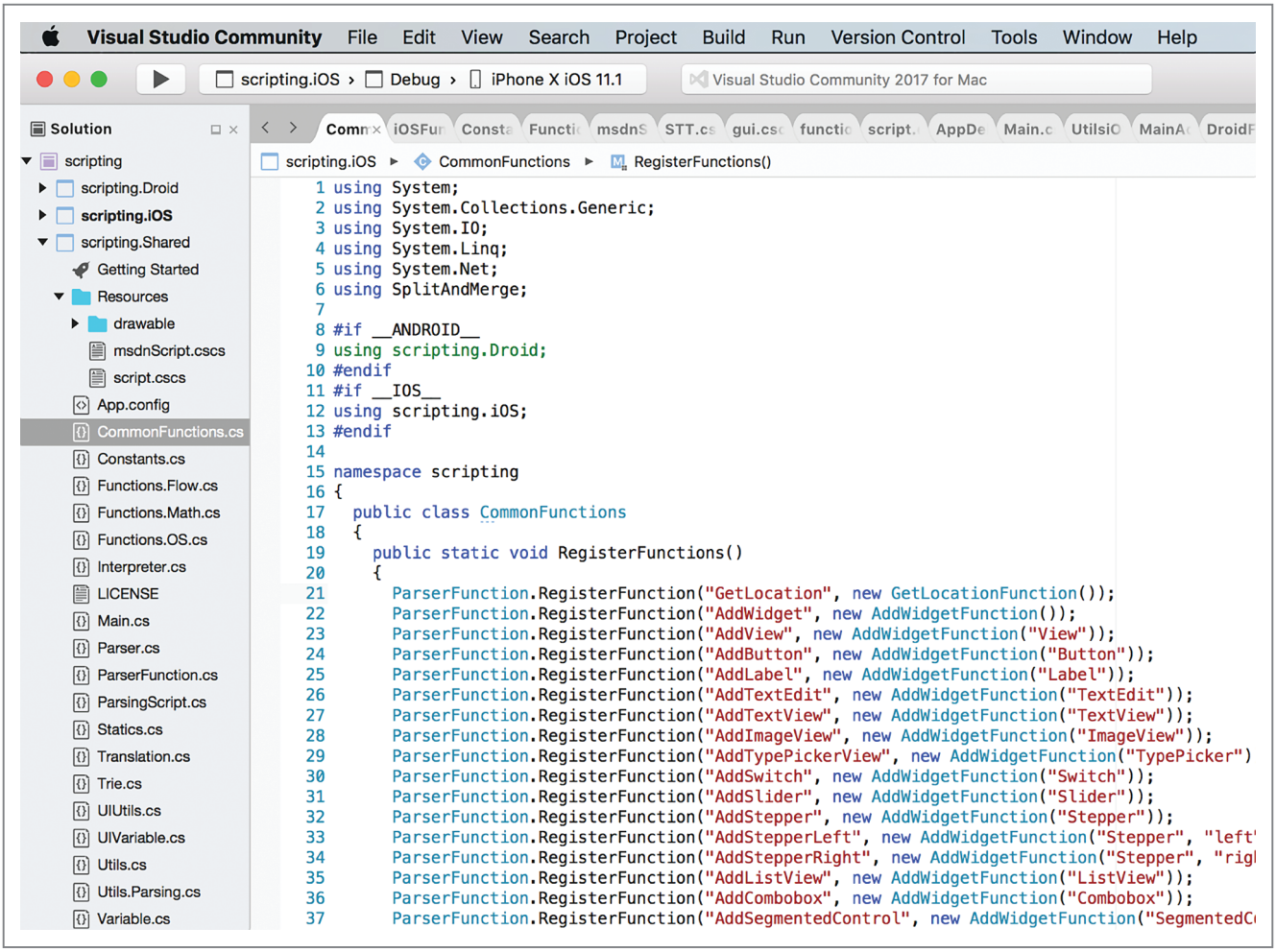


Figure 3 **General Structure of a Xamarin Project with CSCS Scripting**

General Structure of the Project

Where in the workflow will the CSCS code be executed? The answer is different for the iOS and Android projects. You'll see it in what follows, but the full details are in the accompanying source code download at github.com/vassilych/mobile.

The common code, used by both platforms, is in the shared project part, scripting.Shared, which contains all the C# files needed for parsing the CSCS code. The code specific to each platform is located in the scripting.iOS and scripting.Droid projects. See the structure of a sample project in **Figure 3**.

The actual CSCS script is located in the msdnScript.cscs file under the Resources folder in the scripting.Shared project. Note that you can include other CSCS files by calling the following CSCS function:

```
ImportFile("anotherScriptFile.cscs");
```

For the Android project I set up a link to the msdnScript.cscs file from the scripting.Droid Assets folder, and for the iOS project I set up a link from the scripting.iOS Resources folder. You can also reference the script in a number of ways, for example keeping different versions of the script on different platforms.

The CommonFunctions.cs file contains functionality common to iOS and Android. In particular, it holds the method that executes the msdnScripting.cscs script that's shown in **Figure 4**. Note that I distinguish between the iOS- and Android-specific code by using the preprocessor __IOS__ and __ANDROID__ directives. The platform-specific code is mostly located in the corresponding projects, scripting.iOS or scripting.Droid.

Where do you call the RunScript function from? You can call it only after the global layout has been initialized, so you can add widgets to it.

It turns out that it's trickier to do this on Android than on iOS: Just calling the RunScript function at the end of the MainActivity.OnCreate function fails because some variables haven't been initialized yet. So you must put RunScript right before the main activity actually starts running. The Android Activity Lifestyle documentation at goo.gl/yF8dTZ provides a clue: It must go right after the MainActivity.OnResume method completes. Some global variables (for instance, the screen size, the orientation and so on) are not yet initialized even at the end of the OnResume method, so the trick is to register a global layout watcher at the end of the OnResume method that will be triggered as soon as the global layout is constructed:

```
protected override void OnResume()
{
  base.OnResume();
  if (!m_scriptRun) {
    ViewTreeObserver vto = TheLayout.ViewTreeObserver;
    vto.AddOnGlobalLayoutListener(new LayoutListener());
    m_scriptRun = true;
  }
}
```

Where do you call the RunScript function from? You can call it only after the global layout has been initialized, so you can add widgets to it.

Note that I use a special Boolean variable m_scriptRun to make sure that the script runs just once. The OnGlobalLayout method in the layout listener then executes the script:

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

For iOS the situation is somewhat easier, you can just run the script at the end of the AppDelegate.FinishedLaunching method.

Figure 4 **Executing the CSCS Script**

```
public static void RunScript()
{
  RegisterFunctions();
  string fileName = "msdnScript.cscs";
  string script = "";

#if __ANDROID__
  Android.Content.Res.AssetManager assets = MainActivity.TheView.Assets;
  using (StreamReader sr = new StreamReader(assets.Open(fileName))) {
    script = sr.ReadToEnd();
  }
#endif
#if __IOS__
  string[] lines = System.IO.File.ReadAllLines(fileName);
  script = string.Join("\n", lines);
#endif

  Variable result = null;
  try {
    result = Interpreter.Instance.Process(script);
  } catch (Exception exc) {
    Console.WriteLine("Exception: " + exc.Message);
    Console.WriteLine(exc.StackTrace);
    ParserFunction.InvalidateStacksAfterLevel(0);
    throw;
  }
}
```

Text-to-Speech

Let's see how to add some functionality to CSCS, using text-to-speech as an example.

Figure 5 **Speak Function Implementation**

```
public class SpeakFunction : ParserFunction
{
  protected override Variable Evaluate(ParsingScript script)
  {
    bool isList = false;
    List<Variable> args = Utils.GetArgs(script,
        Constants.START_ARG, Constants.END_ARG, out isList);
    Utils.CheckArgs(args.Count, 1, m_name);

    TTS.Init();

    string phrase = args[0].AsString();
    TTS.Voice     = Utils.GetSafeString(args, 1, TTS.Voice);

    TTS.Speak(phrase);

    return Variable.EmptyInstance;
  }
}
```

AUGUST 6 – 10, 2018 • Microsoft Headquarters, Redmond, WA

Change is ~~Coming~~. HERE

Join us for TechMentor, August 6 – 8, 2018, as we return to Microsoft Headquarters in Redmond, WA. In today's IT world, more things change than stay the same. As we celebrate the 20th year of TechMentor, we are more committed than ever to providing immediately usable IT education, with the tools you need today, while preparing you for tomorrow – **keep up, stay ahead and avoid Winter, ahem, Change.**

Plus you'll be at the source, Microsoft HQ, where you can have lunch with Blue Badges, visit the company store, and experience life on campus for a week!

Are You Ready?

Hot Topics Covered:

- Windows PowerShell
- Windows Server
- DevOps
- Azure
- Hyper-V
- IT Security

EVENT PARTNER Microsoft

SUPPORTED BY Redmond · Redmond Channel Partner · Virtualization & Cloud Review

PRODUCED BY 1105 MEDIA — YOUR GROWTH, OUR BUSINESS.

In-Depth Training for IT Pros @ Microsoft Headquarters

You owe it to yourself, your company and your career to be at TechMentor Redmond 2018!

SAVE $400!
REGISTER NOW
Use Promo Code TMJAN2

techmentorevents.com/redmond

CONNECT WITH TECHMENTOR

Twitter
@TechMentorEvent

Facebook
Search "TechMentor"

LinkedIn
Search "TechMentor"

First, I need to create a class deriving from the ParserFunction class and override its protected virtual Evaluate method, as shown in **Figure 5**.

This class is just a wrapper over the actual text-to-speech implementation. For iOS, the text-to-speech implementation is shown in **Figure 6**. The Android implementation is similar, but it takes a bit more coding. You can see it in the accompanying source code download.

> Once I have an implementation, I need to plug it in to the parser.

Once I have an implementation, I need to plug it in to the parser. This is done in the shared project in CommonFunctions.RegisterFunctions static method (also shown in **Figure 3**):

```
ParserFunction.RegisterFunction("Speak", new SpeakFunction());
```

Voice Recognition

For voice recognition I need to use a callback function in order to tell the user what word was actually recognized (or to report an error, as in **Figure 2**).

Figure 6 **iOS Text-to-Speech Implementation (Fragment)**

```
using AVFoundation;

namespace scripting.iOS
{
  public class TTS
  {
    static AVSpeechSynthesizer g_synthesizer = new AVSpeechSynthesizer();
    static public float  SpeechRate { set; get; }      = 0.5f;
    static public float  Volume { set; get; }          = 0.7f;
    static public float  PitchMultiplier { set; get; } = 1.0f;
    static public string Voice { set; get; }           = "en-US";
    static bool m_initDone;

    public static void Init()
    {
      if (m_initDone) {
        return;
      }
      m_initDone = true;
      // Set the audio session category, then it will speak
      // even if the mute switch is on.
      AVAudioSession.SharedInstance().Init();
      AVAudioSession.SharedInstance().SetCategory(AVAudioSessionCategory.Playback,
        AVAudioSessionCategoryOptions.DefaultToSpeaker);
    }

    public static void Speak(string text)
    {
      if (g_synthesizer.Speaking) {
        g_synthesizer.StopSpeaking(AVSpeechBoundary.Immediate);
      }

      var speechUtterance = new AVSpeechUtterance(text) {
        Rate = SpeechRate * AVSpeechUtterance.MaximumSpeechRate,
        Voice = AVSpeechSynthesisVoice.FromLanguage(Voice),
        Volume = Volume,
        PitchMultiplier = PitchMultiplier
      };

      g_synthesizer.SpeakUtterance(speechUtterance);
    }
  }
}
```

I'm going to implement two functions for voice recognition—one to start voice recognition and another to cancel it. These two functions are registered with the parser just as I registered text-to-speech in the previous section:

```
ParserFunction.RegisterFunction("VoiceRecognition", new VoiceFunction());
ParserFunction.RegisterFunction("StopVoiceRecognition", new StopVoiceFunction());
```

The implementation of these two functions for iOS is shown in **Figure 7**. For Android the implementation is similar, but note that voice recognition was added to iOS only in version 10.0, so I must check the device version and, if necessary, inform the user that the device doesn't support it in iOS versions prior to 10.0.

The actual voice recognition code is in the SST class. It's a bit too long to show here and is also different for iOS and Android. I invite you to check it out in the accompanying source code.

Figure 7 **Voice Recognition Implementation**

```
public class VoiceFunction : ParserFunction
{
  static STT m_speech = null;
  public static  STT LastRecording { get { return m_speech; }}

  protected override Variable Evaluate(ParsingScript script)
  {
    bool isList = false;
    List<Variable> args = Utils.GetArgs(script,
                          Constants.START_ARG, Constants.END_ARG, out isList);
    Utils.CheckArgs(args.Count, 1, m_name);

    string strAction = args[0].AsString();
    STT.Voice = Utils.GetSafeString(args, 1, STT.Voice).Replace('_', '-');

    bool speechEnabled = UIDevice.CurrentDevice.CheckSystemVersion(10, 0);
    if (!speechEnabled) {
      UIVariable.GetAction(strAction, "\"" +
       string.Format("Speech recognition requires iOS 10.0 or higher.
       You have iOS {0}",
                     UIDevice.CurrentDevice.SystemVersion) + "\"", "");
      return Variable.EmptyInstance;
    }

    if (!STT.Init()) {
      // The user didn't authorize accessing the microphone.
      return Variable.EmptyInstance;
    }

    UIViewController controller = AppDelegate.GetCurrentController();
    m_speech = new STT(controller);
    m_speech.SpeechError += (errorStr) => {
      Console.WriteLine(errorStr);
      controller.InvokeOnMainThread(() => {
        UIVariable.GetAction(strAction, "\"" + errorStr + "\"", "");
      });
    };
    m_speech.SpeechOK += (recognized) => {
      Console.WriteLine("Recognized: " + recognized);
      controller.InvokeOnMainThread(() => {
        UIVariable.GetAction(strAction, "", "\"" + recognized + "\"");
      });
    };

    m_speech.StartRecording(STT.Voice);

    return Variable.EmptyInstance;
  }
}
public class StopVoiceFunction : ParserFunction
{
  protected override Variable Evaluate(ParsingScript script)
  {
    VoiceFunction.LastRecording?.StopRecording();
    script.MoveForwardIf(Constants.END_ARG);
    return Variable.EmptyInstance;
  }
}
```

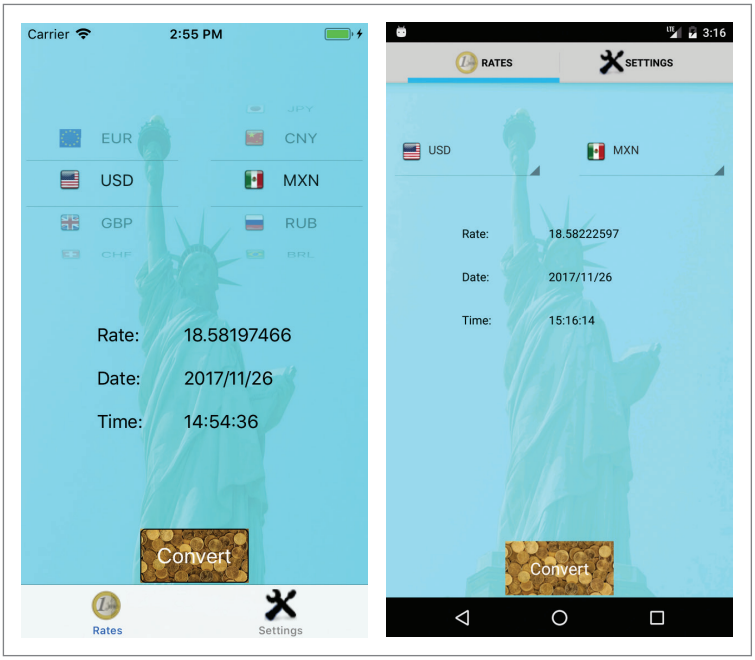


Figure 8 **Currency Convertor in Portrait Mode on iPhone (Left) and on Android (Right)**

Functions on_about and on_refresh are both callbacks that happen when the user clicks on a button.

The on_about method is executed when the user clicks on the "Powered by" button in the Settings tab, which causes the OpenUrl function to open the exchangerate-api.com homepage in the default browser (this tab isn't shown in **Figure 8** and **Figure 9**). The on_refresh method is executed when the user clicks on the Convert button. You then get the selected currencies and the CSCS currency_request function is invoked, which does the actual rate conversion.

To get up-to-date exchange rates in your app, you have to use an online service. I chose exchangerate-api.com.

I didn't have a callback function in text-to-speech, but you can add one in a similar way to tell the user when the speech completes (or if there's an error). The callback to the CSCS code is performed by invoking the UIVariable.GetAction method:

```
public static Variable GetAction(string funcName, string
senderName, string eventArg)
{
  if (senderName == "") {
    senderName = "\"\"";
  }
  if (eventArg == "") {
    eventArg = "\"\"";
  }
  string body = string.Format("{0}({1},{2});", funcName,
senderName, eventArg);

  ParsingScript tempScript = new ParsingScript(body);
  Variable result = tempScript.ExecuteTo();

  return result;
}
```

You can see how this function is used in **Figure 7**.

Example: A Currency Convertor

As an example of using different CSCS features for cross-platform app development, let's create an app from scratch—a currency convertor.

To get up-to-date exchange rates in your app, you have to use an online service. I chose exchangerate-api.com. The site provides an easy-to-use Web service where the first 1,000 requests per month are free, which should be enough to start. After registration you get a unique key you must supply with each request.

My app has different views in portrait and landscape modes. **Figure 8** shows portrait mode and **Figure 9** shows landscape mode for both iPhone and Android.

Figure 10 contains the entire CSCS implementation of the currency convertor app.

The currency_request function first checks if both currencies are the same—in this case I already know that the rate is 1 and there's no need to call a Web service (I want to save my free limited uses

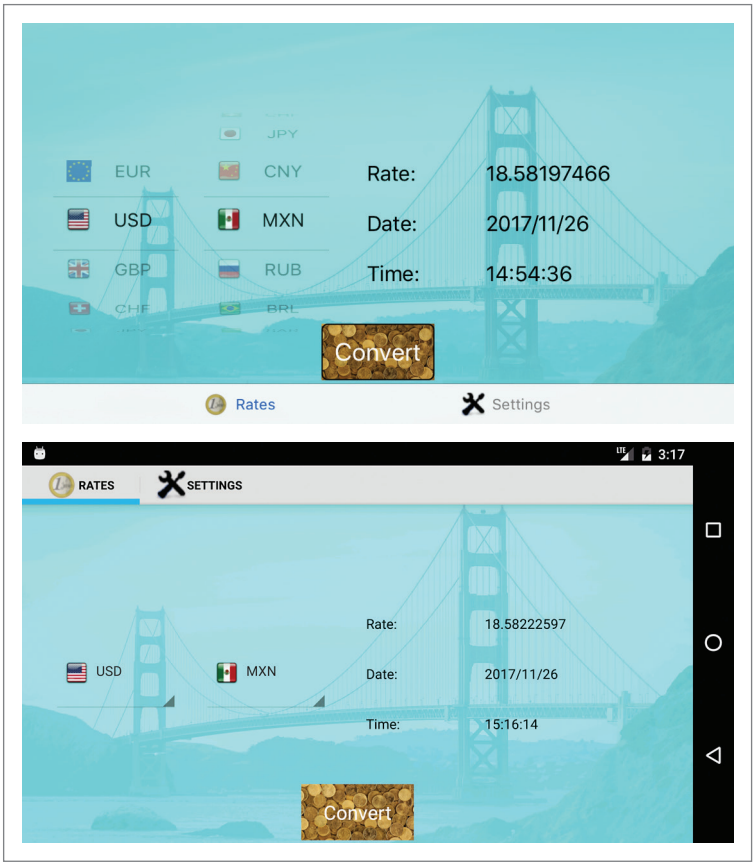


Figure 9 **Currency Convertor in Landscape Mode on iPhone (Top) and on Android (Bottom)**

of this service per month). Otherwise, the WebRequest function is called. This function is common to both iOS and for Android and its implementation is shown in **Figure 11**. Note that you don't have to do the exception handling in C# code. If an exception is thrown (for example, if the service is unavailable), the exception will be propagated to the CSCS code, where it will be caught. Note

Figure 10 **CSCS Implementation of the Currency Convertor App**

```
function on_about(sender, arg) {
  OpenUrl("http://www.exchangerate-api.com");
}

function on_refresh(sender, arg) {
  currency1 = GetText(cbCurrency1);
  currency2 = GetText(cbCurrency2);
  currency_request(currency1, currency2);
}

function currency_request(currency1, currency2) {
  if (currency1 == currency2) {
    time = Now("HH:mm:ss");
    date = Now("yyyy/MM/dd");
    rate = 1;
  } else {
    url = apiUrl + currency1 + "/" + currency2;
    try {
      data = WebRequest(url);
    } catch(exception) {
      WriteConsole(exception.Stack);
      ShowToast("Couldn't get rates. " + exception);
      SetText(labelRateValue, "Error");
      return;
    }

    try {
      timestamp = StrBetween(data, "\"timestamp\":", ",");
      time      = Timestamp(timestamp, "HH:mm:ss");
      date      = Timestamp(timestamp, "yyyy/MM/dd");
      rate      = StrBetween(data, "\"rate\":", "}");
    } catch(exception) {
      ShowToast("Couldn't get rates. " + exception);
      SetText(labelRateValue, "Error");
      return;
    }
  }

  SetText(labelRateValue, rate);
  SetText(labelDateValue, date);
  SetText(labelTimeValue, time);
}

function init() {
  currencies = {"EUR", "USD", "GBP", "CHF", "JPY", "CNY", "MXN", "RUB", "BRL", "SAR"};
  flags      = {"eu_EU", "en_US", "en_GB", "de_CH", "ja_JP", "zh_CN",
                "es_MX", "ru_RU", "pt_BR", "ar_SA"};

  AddWidgetData(cbCurrency1, currencies);
  AddWidgetImages(cbCurrency1, flags);
  SetSize(cbCurrency1, 80, 40);
  SetText(cbCurrency1, "USD");

  AddWidgetData(cbCurrency2, currencies);
  AddWidgetImages(cbCurrency2, flags);
  SetSize(cbCurrency2, 80, 40);
  SetText(cbCurrency2, "MXN");

  SetImage(buttonRefresh,     "coins");
  AddAction(buttonRefresh,    "on_refresh");
  SetFontColor(buttonRefresh, "white");
  SetFontSize(buttonRefresh,  20);

  AddAction(aboutButton,      "on_about");
}

function on_portrait(sender, arg) {
  AddOrSelectTab("Rates", "rates_active.png", "rates_inactive.png");
  SetBackground("us_bg.png");

  locCurrency1 = GetLocation("ROOT", "LEFT", "ROOT", "TOP", 10, 80);
  AddCombobox(locCurrency1, "cbCurrency1", "", 280, 100);

  locCurrency2 = GetLocation("ROOT", "RIGHT", cbCurrency1, "CENTER", -10);
  AddCombobox(locCurrency2, "cbCurrency2", "", 280, 100);

  locRateLabel = GetLocation("ROOT", "CENTER", cbCurrency2, "BOTTOM", -80, 60);
  AddLabel(locRateLabel, "labelRate", "Rate:", 200, 80);

  locRateValue = GetLocation("ROOT", "CENTER", labelRate, "CENTER", 100);
  AddLabel(locRateValue, "labelRateValue", "", 240, 80);

  locDateLabel = GetLocation("ROOT", "CENTER", labelRate, "BOTTOM", -80);
  AddLabel(locDateLabel, "labelDate", "Date:", 200, 80);

  locDateValue = GetLocation("ROOT", "CENTER", labelDate, "CENTER", 100);
  AddLabel(locDateValue, "labelDateValue", "", 240, 80);

  locTimeLabel = GetLocation("ROOT", "CENTER", labelDate, "BOTTOM", -80);
  AddLabel(locTimeLabel, "labelTime", "Time:", 200, 80);

  locTimeValue = GetLocation("ROOT", "CENTER", labelTime, "CENTER", 100);
  AddLabel(locTimeValue, "labelTimeValue", "", 240, 80);

  locRefresh = GetLocation("ROOT", "CENTER", "ROOT", "BOTTOM", 0, -4);
  AddButton(locRefresh, "buttonRefresh", "Convert", 200, 100);

  AddOrSelectTab("Settings", "settings_active.png", "settings_inactive.png");
  locAbout = GetLocation("ROOT", "CENTER", "ROOT", "BOTTOM", -4);
  AddButton(locAbout, "aboutButton", "Powered by exchangerate-api.com", 360, 100);
}

function on_landscape(sender, arg) {
  AddOrSelectTab("Rates", "rates_active.png", "rates_inactive.png");
  SetBackground("us_w_bg.png");

  locCurrency1 = GetLocation("ROOT", "LEFT", "ROOT", "CENTER", 50);
  AddCombobox(locCurrency1, "cbCurrency1", "", 200, 120);

  locCurrency2 = GetLocation(cbCurrency1, "RIGHT", "ROOT", "CENTER", 40);
  AddCombobox(locCurrency2, "cbCurrency2", "", 200, 120);

  locDateLabel = GetLocation(cbCurrency2, "RIGHT", "ROOT", "CENTER", 60);
  AddLabel(locDateLabel, "labelDate", "Date:", 180, 80);

  locDateValue = GetLocation(labelDate, "RIGHT", labelDate, "CENTER", 10);
  AddLabel(locDateValue, "labelDateValue", "", 220, 80);

  locRateLabel = GetLocation(cbCurrency2, "RIGHT", labelDate, "TOP", 60);
  AddLabel(locRateLabel, "labelRate", "Rate:", 180, 80);

  locRateValue = GetLocation(labelRate, "RIGHT", labelRate, "CENTER", 10);
  AddLabel(locRateValue, "labelRateValue", "", 220, 80);

  locTimeLabel = GetLocation(cbCurrency2, "RIGHT", labelDate, "BOTTOM", 60);
  AddLabel(locTimeLabel, "labelTime", "Time:", 180, 80);

  locTimeValue = GetLocation(labelTime, "RIGHT", labelTime, "CENTER", 10);
  AddLabel(locTimeValue, "labelTimeValue", "", 220, 80);

  locRefresh = GetLocation("ROOT", "CENTER", "ROOT", "BOTTOM", 0, -4);
  AddButton(locRefresh, "buttonRefresh", "Convert", 180, 90);

  AddOrSelectTab("Settings", "settings_active.png", "settings_inactive.png");
  locAbout = GetLocation("ROOT", "CENTER", "ROOT", "BOTTOM", -4);
  AddButton(locAbout, "aboutButton", "Powered by exchangerate-api.com", 360, 100);
}

AutoScale();
apiUrl = "https://v3.exchangerate-api.com/pair/c2cd68c6d7b852231b6d69ee/";

RegisterOrientationChange("on_portrait", "on_landscape");
init();

if (Orientation == "Portrait") {
  on_portrait("", "");
} else {
  on_landscape("", "");
}

SelectTab(0);
```

Spreadsheets Everywhere.

SpreadsheetGear 2017 Released

SpreadsheetGear 2017 adds a new SpreadsheetGear for .NET Standard product, official support for Excel 2013 and Excel 2016, 51 new Excel functions for a total of 449 fully supported functions, full conditional formatting support, enhanced workbook protection and encryption, cell gradient rendering and more.

Support for iOS, Android, Linux, macOS, UWP and more

SpreadsheetGear for .NET Standard enables cross-platform developers to enjoy the same high performance Excel-compatible reporting, charting, calculations and more relied on by thousands of Windows developers for 10+ years.

Scalable Reporting

Easily create richly formatted Excel reports without Excel from any ASP.NET, Windows Forms, WPF or Silverlight application using spreadsheet technology built from the ground up for performance, scalability and reliability.

WIN Windows Forms | Silverlight | WPF

Powerful Controls

Add powerful Excel-compatible viewing, editing, formatting, calculating, filtering, sorting, charting, printing and more to your WinForms, WPF and Silverlight applications.

Comprehensive Charting

Enable users to visualize data with comprehensive Excel-compatible charting which makes creating, modifying, rendering and interacting with complex charts easier than ever before.

Fastest Calculations

Evaluate complex Excel-based models and business rules with the fastest and most complete Excel-compatible calculation engine available.

Download your free fully functional evaluation at SpreadsheetGear.com

SpreadsheetGear

Toll Free USA (888) 774-3273 | Phone (913) 390-4797 | sales@spreadsheetgear.com

Figure 11 **C# Implementation of the WebRequestFunction Evaluate Method**

```
public class WebRequestFunction : ParserFunction
{
  protected override Variable Evaluate(ParsingScript script)
  {
    bool isList = false;
    List<Variable> args = Utils.GetArgs(script,
                          Constants.START_ARG, Constants.END_ARG, out isList);
    Utils.CheckArgs(args.Count, 1, m_name);
    string uri = args[0].AsString();

    string responseFromServer = "";
    WebRequest request = WebRequest.Create(uri);

    using (WebResponse response = request.GetResponse()) {
      Console.WriteLine("{0} status: {1}", uri,
                        ((HttpWebResponse)response).StatusDescription);
      using (StreamReader sr = new StreamReader(response.GetResponseStream())) {
        responseFromServer = sr.ReadToEnd();
      }
    }

    return new Variable(responseFromServer);
  }
}
```

also that the WebRequest function is implemented synchronously. You can also make it asynchronous by supplying the callback function to be called when the request is done (analogous to the voice recognition functionality I showed earlier).

Let's continue analyzing the CSCS code in **Figure 10**. I was describing what happens in the currency_request function. The JSON response I get from exchangerate-api.com looks like the following:

```
{"result":"success","timestamp":1511464063,"from":"USD","to":"CHF","rate":0.99045395}
```

> If an exception is thrown (for example, if the service is unavailable), the exception will be propagated to the CSCS code, where it will be caught.

The timestamp is the number of seconds passed since Jan. 1, 1970. The CSCS function Timestamp(format) converts this number of seconds to a specified date or time format.

StrBetween(data, strStart, strEnd) is a convenience function for extracting a substring from the data string between strStart1 and strStart2 strings.

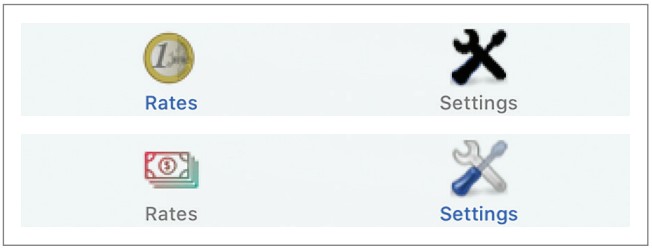


Figure 12 **Active and Inactive Tabs on iOS**

Once I extract the rate, date and time, I set them to the corresponding labels using the SetText(widgetName, text) function.

In the init function I initialize the data, and I can add additional currencies for the conversion.

It's easy to have different layouts for different orientations: register orientation change callbacks with the RegisterOrientationChange function. The on_portrait and on_landscape functions are called each time the device orientation changes. As you can see at the bottom of **Figure 10**, it's set up by invoking:

```
RegisterOrientationChange("on_portrait", "on_landscape");
```

In general you add widgets to the screen at particular locations, using the logic explained in the "Hello, World!" example in the first section. You probably noticed the different phone backgrounds for landscape and portrait mode. This is done using the SetBackground(imageName) CSCS function.

AddOrSelectTab function has the following signature:

```
AddOrSelectTab(Tab_Name, Picture_when_active, Picture_when_inactive);
```

If the the tab doesn't exist yet, it will be added, otherwise it will be selected and all consecutive widgets will be added to this tab. **Figure 12** shows how the tabs look in both active and inactive modes.

Wrapping Up

In this article you saw that with CSCS you can program mobile apps using a scripting language. The script is converted to native code using the C# interpreter and the Xamarin Framework. The CSCS scripts can do anything that can be done in C# (and in Xamarin C# you can do anything that can be done in native app development).

I've already published an app written entirely in CSCS. Check out the iOS version at apple.co/2yixGxZ and the Android version at goo.gl/zADtNb.

CSCS scripting for mobile apps is far from complete. To add new functionality to CSCS, you create a new class that derives from the ParserFunction class and override its Evaluate method. Then you register that class with the parser, supplying its CSCS name:

```
ParserFunction.RegisterFunction("CSCS_Name", new MyNewCustomFunction())
```

Using CSCS you can place all the widgets programmatically, and the same code will be used for both Android and iOS. And you don't need to use any XAML for that, as you would with Xamarin.Forms.

You can also combine CSCS with the existing C# code—it's easy to call C# code from CSCS, as I explained at codemag.com/article/1711081. In that article you can also check the list of functions implemented in CSCS. But for the latest, most up-to-date CSCS functions and features, visit github.com/vassilych/mobile.

Unfortunately, there's no room to discuss some other cool things you can do in CSCS, such as in-app purchasing and billing, in-app advertisements, scheduling one-shot and repetitive events, and more, but you can check them out in the accompanying source code download. ■

Vassili Kaplan *is a former Microsoft Lync developer. He is passionate about programming in C#, C++, Python and now in CSCS. He currently lives in Zurich, Switzerland, and works as a freelancer for various banks. You can reach him at iLanguage.ch*

Thanks to the following Microsoft technical expert for reviewing this article: *James McCaffrey*

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

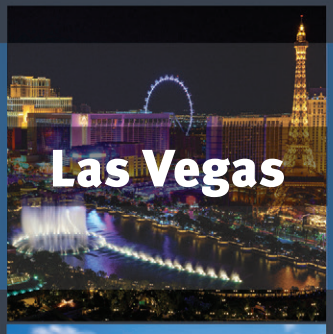
Visual Studio LIVE! 25 YEARS OF CODING INNOVATION

VISUAL STUDIO LIVE! (VSLive!™) is celebrating 25 years as one of the most respected, longest-standing, independent developer conferences, and we want you to be a part of it. Join us in 2018 for #VSLive25, as we highlight how far technology has come in 25 years, while looking toward the future with our unique brand of training on .NET, the Microsoft Platform and open source technologies in seven great cities across the US.

HELP US CELEBRATE #VSLIVE25. WHICH LOCATION WILL YOU ATTEND IN 2018?

MARCH 11 – 16
Bally's Hotel & Casino

Respect the Past. Code the Future.



APRIL 30 – MAY 4
Hyatt Regency Austin

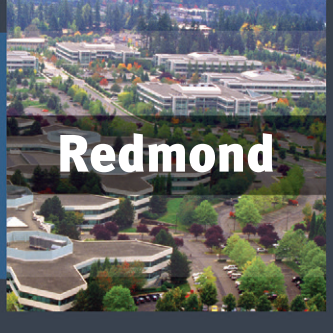
Code Like It's 2018!

JUNE 10 – 14
Hyatt Regency Cambridge

Developing Perspective.

AUGUST 13 – 17
Microsoft Headquarters

Yesterday's Knowledge; Tomorrow's Code!



SEPTEMBER 17 – 20
Renaissance Chicago

Look Back to Code Forward.

OCTOBER 7 – 11
Hilton San Diego Resort

Code Again for the First Time!

DECEMBER 2 – 7
Loews Royal Pacific Resort

Code Odyssey.

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

linkedin.com – Join the "Visual Studio Live" group!

SUPPORTED BY

Microsoft | Visual Studio | msdn magazine | Visual Studio MAGAZINE

PRODUCED BY

1105 MEDIA YOUR GROWTH. OUR BUSINESS.

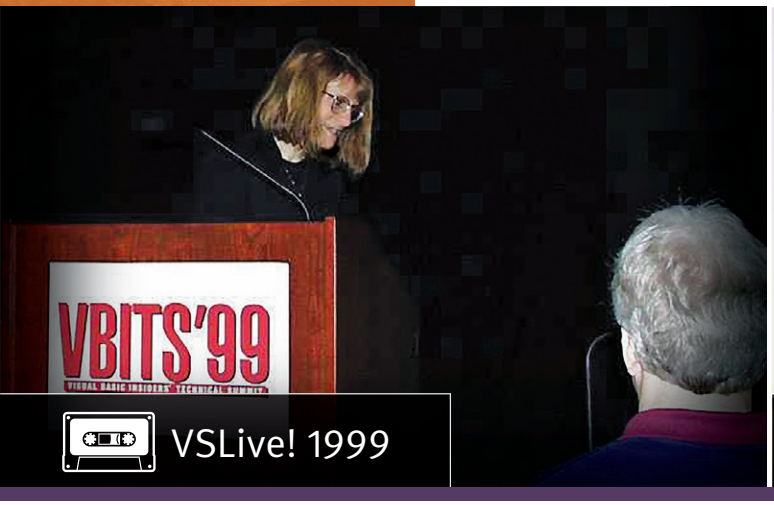
vslive.com

#VSLIVE25

We're Gonna Code Like It's 2018

Visual Studio Live! (VSLive!™) is back in Austin this spring with 5 days of practical, unbiased, Developer training. From April 30 – May 4, 2018, join us for hard-hitting sessions, insightful workshops, intense hands-on labs and fun networking events. Code with industry experts, hear the latest from Microsoft insiders and tune up on today's hottest training topics! Plus, help us celebrate 25 years of coding innovation as we take a fun look back at technology and training since 1993. Come experience the education, knowledge-share and networking at #VSLive25.

Visual Studio LIVE! 25 YEARS OF CODING INNOVATION



SUPPORTED BY

Microsoft · Visual Studio · msdn magazine · Visual Studio MAGAZINE

PRODUCED BY

1105 MEDIA — YOUR GROWTH. OUR BUSINESS.

DEVELOPMENT TOPICS INCLUDE:

- VS2017/.NET
- JavaScript / Angular
- Xamarin
- Software Practices
- Database & Analytics
- ASP.NET / Web Server
- ALM / DevOps
- Azure / Cloud
- UWP (Windows)
- Hands-On Labs — BACK BY POPULAR DEMAND

"I have been looking forward to attending the event for some years and I was finally able to come. I need to help keep my company current and secure. I am advocating for more test-driven dev and using more async as a resulting of attending the conference!"

– Yessenia Figueroa, 1-800-Contacts

"This is my second year attending Visual Studio Live! I want to come back year over year because the variety of sessions allow me to deep dive into tech that I use on a daily basis, but the option is there to explore something new. The biggest dev improvement I made last year was from the great hands-on experience I got with SignalR at VSLive!"

– Jake Clauson, Washington Technology Solutions

Register to code with us today!

Register Now and Save $300!

Use promo code VSLJAN2

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

linkedin.com – Join the "Visual Studio Live" group!

vslive.com/austinmsdn

Developing Perspective

Visual Studio Live! (VSLive!™) returns to Boston, June 10 – 14, 2018, with 5 days of practical, unbiased, Developer training, including NEW intense hands-on labs. Join us as we dig into the latest features of Visual Studio 2017, ASP.NET Core, Angular, Xamarin, UWP and more. Code with industry experts AND Microsoft insiders while developing perspective on the Microsoft platform. Plus, help us celebrate 25 years of coding innovation as we take a fun look back at technology and training since 1993. Come experience the education, knowledge-share and networking at #VSLive25.

Visual Studio LIVE! 25 YEARS OF CODING INNOVATION

SUPPORTED BY

Microsoft Visual Studio msdn magazine Visual Studio MAGAZINE

PRODUCED BY

1105 MEDIA YOUR GROWTH. OUR BUSINESS.

DEVELOPMENT TOPICS INCLUDE:

- VS2017/.NET
- JavaScript / Angular
- Xamarin
- Software Practices
- Database & Analytics
- ASP.NET / Web Server
- ALM / DevOps
- Azure / Cloud
- UWP (Windows)
- Hands-On Labs (NEW!)

"I have attended 12+ Visual Studio Live events, starting with the early days of VBITS and C# Live. I return every year because no one reflects and enhances what is currently happening in and with Microsoft's developer world like VSLive! I honestly trust and enjoy all the speakers I look to them as both mentors and old friends!"

– John Kasarda, Accuquote

"Last year, I got so much out of the SOLID principle sessions – it gave me great perspective on the subject. This year, I loved the HoloLens introduction. I work for an aerospace company, and I can see that the technology will be utilized for assembly, 3D design, 3D inspection and many other applications in the near future!"

– Mani Nazari, IT Factory Automation Systems

Register to code with us today!

Register Now and Save $300!

Use promo code VSLJAN2

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

linkedin.com – Join the "Visual Studio Live" group!

vslive.com/bostonmsdn

JAMES MCCAFFREY

Thompson Sampling Using C#

Thompson sampling is an algorithm that can be used to find a solution to a multi-armed bandit problem, a term deriving from the fact that gambling slot machines are informally called "one-armed bandits."

Suppose you're standing in front of three slot machines. When you pull the arm on one of the machines, it will pay you either zero dollars or one dollar according to some probability distribution that's unknown to you. For example, the machine might pay with a mean probability of 0.5, so if you pulled the handle on that machine 100 times, you'd expect to be paid zero dollars about 50 times and one dollar about 50 times.

Your goal is to identify the best machine, as efficiently as possible. The problem just described is an example of a Bernoulli bandit problem because the result is either 0 or 1. There are other types of bandit problems. For example, if each machine paid out a value, usually between zero and one dollar (such as 55.0 cents), according to a bell-shaped Gaussian distribution, you'd have a Gaussian process bandit problem. This article addresses only the Bernoulli bandit problem.

> It's unlikely you'll be asked by your boss to analyze slot machines, but multi-armed bandit problems appear in many important, practical scenarios.

There are several algorithms that can be used to try to find the best machine. Suppose you're limited to a total of 100 pulls on the three machines. You could try 10 pulls on each machine, keeping track of how well each machine performed. Then you could use your remaining 70 pulls on the one machine that paid out the most money during your 30-pull explore phase. The danger with this approach is that you might incorrectly identify the best machine. And if you use many pulls during the explore phase, you won't have many left during the exploit phase.

Code download available at msdn.com/magazine/0218magcode.

Thompson sampling is a clever algorithm that continuously adjusts the estimated probabilities of each machine's payout. As you'll see shortly, the key to Thompson sampling for a Bernoulli bandit problem is the mathematical beta distribution.

It's unlikely you'll be asked by your boss to analyze slot machines, but multi-armed bandit problems appear in many important, practical scenarios. For example, suppose you work for a drug manufacturer. You've just created four new experimental drugs for cancer and you want to establish which of the four drugs is most effective, with a minimum of testing on live subjects. Or perhaps you work for an e-commerce company and you want to determine which of several new advertising campaigns is the most successful.

A good way to see where this article is headed is to take a look at the demo program in **Figure 1**. The demo sets up three Bernoulli machines with probabilities of payout of (0.3, 0.5, 0.7), respectively. In a non-demo scenario, the probabilities are unknown to you, of course. You are allowed a total of 10 pulls. The goal is to determine the best machine (machine #3) and pull its handle the most times.

In the first trial, you assume that all three machines have a payout probability of 0.5. The demo uses the beta distribution to generate three probabilities based on that assumption. These random probabilities are (0.7711, 0.1660, 0.1090). Because the probability associated with machine #1 is highest, its arm is pulled. In this case, machine #1 does not pay out.

In the second trial, behind the scenes, the demo believes that the first machine now has a payout probability that is less than 0.5. Beta sampling is used and this time the probabilities are (0.6696, 0.2250, 0.7654), so machine #3 is selected and played, and its estimated probability of payout is adjusted according to whether the machine pays out or not.

Over time, because machine #3 has the highest probability of payout, it will win more often than the other two machines, and each time machine #3 does pay out, the likelihood that it will be selected increases.

After the 10 trials, machine #1 was played three times and paid out just once so the simulation guesses its true probability of payout is approximately 1/3 = 0.33. Machine #2 was played three times and paid out twice (estimated p = 2/3 = .6667). Machine #3 was played four times and paid out three times (estimated p = 3/4 = 0.7500). So, in this example at least, the best machine was identified and was played the most often.

This article assumes you have intermediate or better programming skills, but doesn't assume you know anything about Thompson sampling or beta distributions. The demo program is coded using

C#, but you shouldn't have too much trouble refactoring the code to another language, such as Visual Basic or Python, if you wish. The code for the demo program is presented in its entirety in this article, and is also available in the accompanying file download.

Understanding the Beta Distribution

Thompson sampling for a Bernoulli bandit problem depends on the beta distribution. In order to understand the beta distribution you must understand probability distributions in general. There are many types of probability distributions, each of which has variations that depend on one or two parameters.

You may be familiar with the uniform distribution, which has two parameters, called min and max, or sometimes just a and b.

```
file:///C:/Thompson/bin/Debug/Thompson.EXE

Begin Thompson sampling demo

Goal is to find best of three machines
Machines pay out with probs 0.3, 0.5, 0.7

Trial    0
sampling probs = 0.7711 0.1660 0.1090
Playing machine 0 -- lose

Trial    1
sampling probs = 0.6696 0.2250 0.7654
Playing machine 2 -- lose

Trial    2
sampling probs = 0.0038 0.3821 0.5421
Playing machine 2 -- win

Trial    3
sampling probs = 0.5764 0.0826 0.3773
Playing machine 0 -- lose

Trial    4
sampling probs = 0.3861 0.0992 0.1762
Playing machine 0 -- win

Trial    5
sampling probs = 0.3022 0.6871 0.6345
Playing machine 1 -- win

Trial    6
sampling probs = 0.0775 0.4168 0.9069
Playing machine 2 -- win

Trial    7
sampling probs = 0.0457 0.5099 0.6876
Playing machine 2 -- win

Trial    8
sampling probs = 0.1125 0.7464 0.2885
Playing machine 1 -- win

Trial    9
sampling probs = 0.1270 0.8496 0.5723
Playing machine 1 -- lose

Final sort-of estimates of machine means:
0.3333  0.6667  0.7500

Number times each machine played:
3  3  4

End demo
```

Figure 1 **Thompson Sampling Demo**

A uniform distribution with min = 0.0 and max = 1.0 will return a p-value between 0.0 and 1.0 where each value is equally likely. Therefore, if you sampled 1,000 times from the uniform distribution, you'd expect to get about 100 p-values between 0.00 and 0.10, about 100 p-values between 0.10 and 0.20, and so on, to about 100 p-values between 0.90 and 1.00. If you graphed the results, you'd see a bar chart with 10 bars, all about the same height.

> There are many types of probability distributions, each of which has variations that depend on one or two parameters.

You might also be familiar with the normal (also called Gaussian) distribution. The normal distribution is also characterized by two parameters, the mean and the standard deviation. If you sampled 1,000 times from the normal distribution with mean = 0.0 and standard deviation = 1.0, you'd expect to get about 380 z-values between -0.5 and +0.5; about 240 z-values between +0.5 and +1.5 (and also between -0.5 and -1.5); about 60 z-values between +1.5 and +2.5 (and also between -1.5 and -2.5); and 10 z-values greater than +2.5 (and 10 less than -2.5). If you graphed the results you'd see a bell-shaped bar chart.

The beta distribution is characterized by two parameters, usually called alpha and beta, or sometimes just a and b. Note the possible confusion between "beta" representing the entire distribution, and "beta," representing the second of the two distribution parameters.

If you sample from a beta distribution with a = 1 and b = 1, you get the exact same results as from the uniform distribution with mean = 0.5. If a and b have different values, when you sample from the beta distribution you get p-values that average to a / (a+b). For example, if a = 3 and b = 1, and you repeatedly sample, you will get

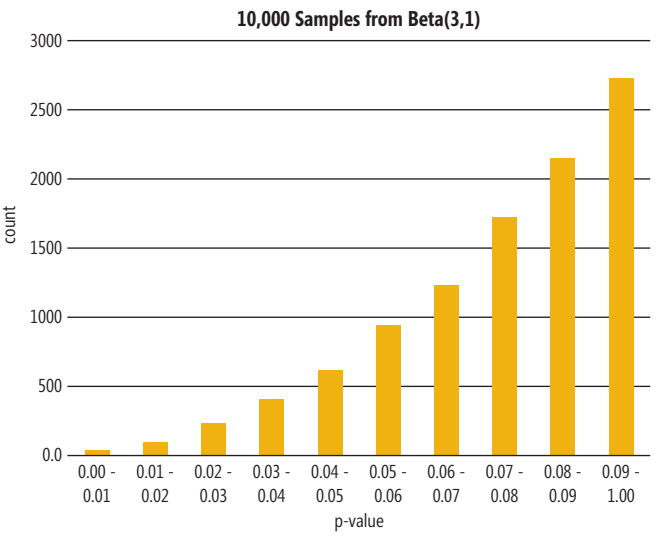


Figure 2 **Sampling from the Beta(3,1) Distribution**

p-values between 0.0 and 1.0, but the average of the p-values will be 3 / (3+1) = 0.75. This means that p-values between 0.90 and 1.00 will be the most common; p-values between 0.80 and 0.90 will be a bit less common; and so on, down to very few p-values between 0.00 and 0.10. The graph in **Figure 2** shows the results of 10,000 samples from the beta distribution with a = 3 and b = 1.

> The demo uses three machines, but Thompson sampling can work with any number of machines.

The connection between the beta distribution and a Bernoulli bandit problem is quite subtle. Briefly, the beta distribution is the conjugate prior for the Bernoulli likelihood function. Although the underlying math is very deep, implementation of the Thompson algorithm is, fortunately, (relatively) simple.

The Demo Program

To create the demo program I launched Visual Studio and created a new console application named Thompson. The demo has no significant .NET Framework dependencies, so any version of Visual Studio is fine. After the template code loaded into the editor window, I right-clicked on file Program.cs and renamed it to the slightly more descriptive ThompsonProgram.cs, and allowed Visual Studio to automatically rename class Program for me. At the top of the template code, I deleted all references to unneeded namespaces, leaving just the reference to the top-level System namespace.

The overall program structure, with a few minor edits to save space, is presented in **Figure 3**. All the control logic is in the Main method. Sampling from the beta distribution is implemented using the program-defined BetaSampler class. All normal error checking is removed to save space.

The demo begins by setting up four arrays:

```
Console.WriteLine("Begin Thompson sampling demo");
int N = 3;
double[] means = new double[] { 0.3, 0.5, 0.7 };
double[] probs = new double[N];
int[] S = new int[N];
int[] F = new int[N];
```

The demo uses three machines, but Thompson sampling can work with any number of machines. The mean probabilities of payouts are hardcoded. The closer the mean probabilities are to each other, the more difficult the problem. The array named probs holds the probabilities from a sampling of the beta distribution, which determine which machine to play. The arrays named S ("success") and F ("failure") hold the cumulative number of times each machine paid out and didn't pay out when played.

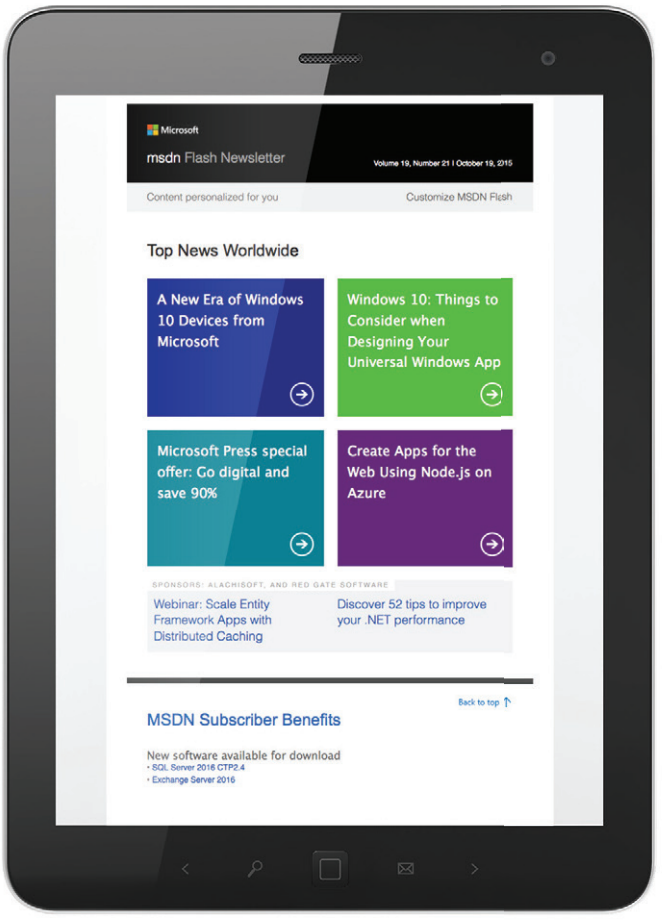
The demo uses two random number-generating objects:

```
Random rnd = new Random(4);
BetaSampler bs = new BetaSampler(2);
```

The rnd object is used to determine whether a selected machine wins or loses. Note that I use the terms win and lose, pay out and not pay out, and success and failure, interchangeably. The bs object is

Figure 3 **Thompson Sampling Demo Program Structure**

```
using System;
namespace Thompson
{
  class ThompsonProgram
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Begin Thompson sampling demo");

      int N = 3;
      double[] means = new double[] { 0.3, 0.5, 0.7 };
      double[] probs = new double[N];
      int[] S = new int[N];
      int[] F = new int[N];

      Random rnd = new Random(4);
      BetaSampler bs = new BetaSampler(2);

      for (int trial = 0; trial < 10; ++trial)
      {
        Console.WriteLine("Trial " + trial);
        for (int i = 0; i < N; ++i)
          probs[i] = bs.Sample(S[i] + 1.0, F[i] + 1.0);

        Console.Write("sampling probs = ");
        for (int i= 0; i < N; ++i)
          Console.Write(probs[i].ToString("F4") + " ");
        Console.WriteLine("");

        int machine = 0;
        double highProb = 0.0;
        for (int i = 0; i < N; ++i) {
          if (probs[i] > highProb) {
            highProb = probs[i];
            machine = i;
          }
        }
        Console.Write("Playing machine " + machine);
        double p = rnd.NextDouble();
        if (p < means[machine]) {
          Console.WriteLine(" -- win");
          ++S[machine];
        }
        else {
          Console.WriteLine(" -- lose");
          ++F[machine];
        }
      }

      Console.WriteLine("Final estimates of means: ");
      for (int i = 0; i < N; ++i)  {
        double u = (S[i] * 1.0) / (S[i] + F[i]);
        Console.WriteLine(u.ToString("F4") + "  ");
      }

      Console.WriteLine("Number times machine played:");
      for (int i = 0; i < N; ++i) {
        int ct = S[i] + F[i];
        Console.WriteLine(ct + "  ");
      }

      Console.WriteLine("End demo ");
      Console.ReadLine();
    }
  }

  public class BetaSampler
  {
    // ...
  }
}
```



Get news from MSDN in your inbox!

Sign up to receive **MSDN FLASH**, which delivers the latest resources, SDKs, downloads, partner offers, security news, and updates on national and local developer events.

msdn magazine

msdn.microsoft.com/flashnewsletter

used to sample the beta distribution. The seeds used, 2 and 4, were specified only because they provided a representative demo run.

The main simulation loop begins:

```
for (int trial = 0; trial < 10; ++trial) {
  Console.WriteLine("Trial " + trial);
  for (int i = 0; i < N; ++i)
    probs[i] = bs.Sample(S[i] + 1.0, F[i] + 1.0);
...
```

You might want to set the number of trials to a large number, like 1,000, to observe how quickly Thompson sampling zeros in on an optimal machine. The key to the entire demo is the call to the Sample method. The numbers of successes and failures are passed to the method. A constant 1.0 is added as something of a hack because the beta distribution requires the a and b parameters to be greater than 0. If you have some prior knowledge of the machines' payout probabilities, you could adjust the constant to reflect that information.

> A quick scan of the code should convince you there's some clever, non-trial math involved.

After displaying the updated sampling probabilities, the demo selects the machine with the greatest sampling probability:

```
int machine = 0;
double highProb = 0.0;
for (int i = 0; i < N; ++i) {
  if (probs[i] > highProb) {
    highProb = probs[i];
    machine = i;
  }
}
```

Because probabilities are sampled, even if a machine has zero wins and many failures, it could still be selected for play. This mechanism enhances the exploration capabilities of the algorithm.

The main iteration loop concludes by playing the selected machine:

```
...
  Console.Write("Playing machine " + machine);
  double p = rnd.NextDouble();
  if (p < means[machine]) {
    Console.WriteLine("win"); ++S[machine];
  }
  else {
    Console.WriteLine("lose"); ++F[machine];
  }
}
```

The NextDouble method returns a uniformly random value between 0.0 and 1.0 and is used to implement a Bernoulli process. The demo concludes by displaying the estimated probabilities of payout for each machine (not bothering to check for possible division by zero), and the number of times each machine was played.

Implementing the Beta Distribution

Somewhat surprisingly, to the best of my knowledge the .NET Framework doesn't have a library method to sample from the beta distribution. Instead of taking a dependency on an external library, I decided to implement one from scratch. There are many algorithms to generate a beta sample. I used the "BA" algorithm, developed by R. C. H. Cheng in 1978. The entire code is presented in **Figure 4**.

Figure 4 **Program-Defined Beta Distribution Sampler**

```
public class BetaSampler
{
  public Random rnd;
  public BetaSampler(int seed)
  {
    this.rnd = new Random(seed);
  }
  public double Sample(double a, double b)
  {
    double alpha = a + b;
    double beta = 0.0;
    double u1, u2, w, v = 0.0;

    if (Math.Min(a, b) <= 1.0)
      beta = Math.Max(1 / a, 1 / b);
    else
      beta = Math.Sqrt((alpha - 2.0) /
(2 * a * b - alpha));
    double gamma = a + 1 / beta;

    while (true) {
      u1 = this.rnd.NextDouble();
      u2 = this.rnd.NextDouble();
      v = beta * Math.Log(u1 / (1 - u1));
      w = a * Math.Exp(v);
      double tmp = Math.Log(alpha / (b + w));
      if (alpha * tmp + (gamma * v) - 1.3862944 >=
 Math.Log(u1 * u1 * u2))
        break;
    }

    double x = w / (b + w);
    return x;
  }
}
```

Sampling from the beta distribution is a fascinating topic in its own right. A quick scan of the code should convince you there's some clever, non-trial math involved. The implementation follows the source research paper closely—the same variables names and so on. Notice the potentially infinite loop, which is common in research pseudo-code, but a definite no-no in production code. You might want to add a loop counter variable and throw an exception if its value exceeds some threshold.

Wrapping Up

This article and its code should give you all the information you need to experiment with Thompson sampling for multi-armed Bernoulli problems. It should also allow you to explore bandit problems with other kinds of payout functions. For example, if you have machines that pay out according to a Poisson likelihood function, instead of using the beta distribution you'd sample from the gamma distribution, which is the conjugate prior for Poisson.

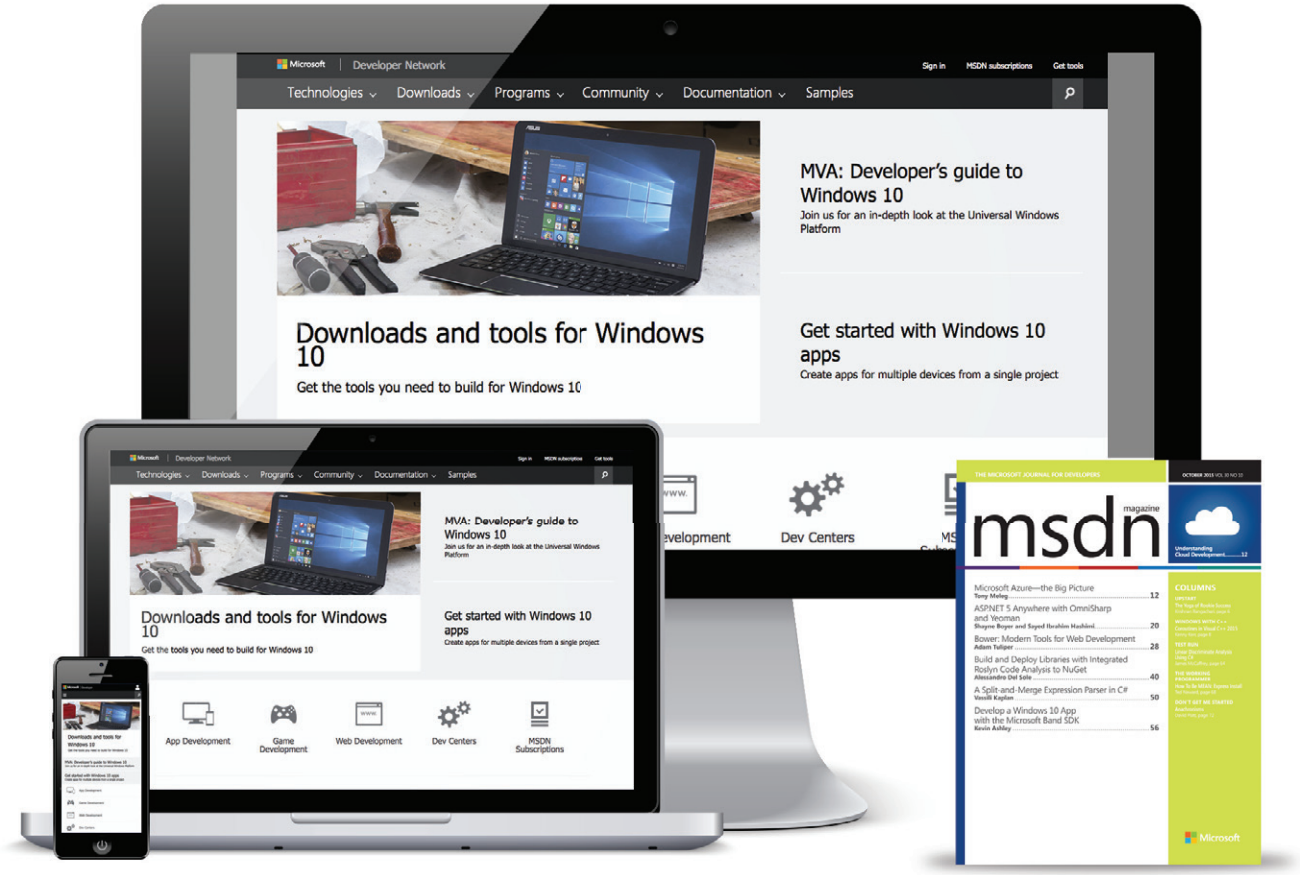
The multi-armed bandit problem is an example of what's called reinforcement learning (RL). In RL machine learning, instead of creating a prediction system using labeled data that has known correct answers, you generate a prediction model on-the-fly, using some sort of reward function. RL is at the forefront of machine learning research. ■

Dr. James McCaffrey *works for Microsoft Research in Redmond, Wash. He has worked on several Microsoft products, including Internet Explorer and Bing. Dr. McCaffrey can be reached at jamccaff@microsoft.com.*

Thanks to the following Microsoft technical experts who reviewed this article: *Chris Lee, Ricky Loynd, Adith Swaminathan*

msdn magazine

Where you need us most.



Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

LIVE! 360
TECH EVENTS WITH PERSPECTIVE

MSDN.microsoft.com

DAVID S. PLATT

Why Software *Still* Sucks

It was 11 years ago that I published my magnum opus, a book titled "Why Software Sucks" (Addison-Wesley, 2006). That title still evokes a chuckle whenever I introduce it. And its principles continue to inspire some readers and infuriate others. I'll begin my ninth year of howling in this column with some thoughts on the book.

I originally got the idea for "Why Software Sucks" (WSS) from Atul Gawande's book, "Complications: A Surgeon's Notes on an Imperfect Science" (Picador, 2002). Nobody can argue that our software applications and their interfaces aren't highly imperfect. And nobody can pick them apart and expose hypocrisy and foolishness like your humble correspondent.

The Amazon reviews of WSS are highly polarized: 42 percent of reviews are five-star, while 27 percent are one-star. More than two-thirds of reviewers hold extreme views on the book—it's great or it stinks—without much in between. Frustrated users are happy to hear an expert say that their software troubles aren't their fault. Many geeks agree, but others are furious that I've ripped the lid off their beloved sausage factory. Both reactions please me immensely.

> What's a prophet to do? What I always do: write another book. Comfort the afflicted and afflict the comfortable.

The software industry has changed so much since I wrote the book—the entire mobile sector has appeared and somewhat matured, for example. The Internet of Things (IoT). Artificial intelligence now busting out all over. Even quantum computing is rapidly approaching, with Microsoft releasing its initial QC toolkit late last year.

And yet I still see today's programs making the same mistake over and over and over: ignoring, misunderstanding or mistreating the user, who's the point of the whole exercise.

The central tenet of WSS holds as true as ever: *Know thy user, for he is not thee*. No matter what you build it with or what platforms you target—desktop or Web, mobile, IoT, or quantum—your software *will always* suck, cannot fail to suck, if you don't actively study who your users are. More important, you need to stop falsely thinking that your users resemble your geeky selves.

If you don't interview actual users to find their true pain points, your software is doomed to suck. You need to test your preliminary designs on users to find what they understand and what they don't. You need to iterate, to make immediate use of what you've learned. You need to use telemetry to discover what users actually do, instead of what they can remember doing, or are willing to admit to doing. (See my video at bit.ly/2kN9E9B for a quick case study of doing this design process well.)

Here's an example that still makes my blood boil. Not long ago, while interviewing the owner of a large customer-facing Web site, I asked about differences in usage patterns between different age groups in the user population. "No difference, really," the owner said. "All ages are equally inept."

I exploded: "It's not your *users' job* to be 'ept!' It is your job—*your job!*—to make the program work, easily and well, with the users you have."

I wish I could lambaste that person here by name and company, but I had promised to keep the interview off-record. My blood pressure rose 20 points just writing that paragraph. I hope yours did, too, while reading it.

What's a prophet to do? What I always do: write another book. Comfort the afflicted and afflict the comfortable. I'm torn between working with a publisher again and self-publishing. I didn't like my last editor: "No, you can't say [that]; you might offend someone." Well, yes, that's the point. I want to offend people who write bad software and can't—or won't—see or acknowledge that it's bad. I want to rub their noses in it publicly to instruct my other readers who are willing to learn from others' bad examples. I don't know how to get the 42 percent five-star reviews without also getting the 27 percent one-star. To anyone who accuses me of calling 'em as I see 'em, I plead guilty as charged. If there's a publisher who wants that, call me.

And so, dear readers, my work is not yet done. Likely, it never will be. But onward. Upward. Outward. The title of the new book will be, can only be: "Why Software *Still* Sucks." ■

DAVID S. PLATT *teaches programming .NET at Harvard University Extension School and at companies all over the world. He's the author of 11 programming books, including "Why Software Sucks" (Addison-Wesley Professional, 2006) and "Introducing Microsoft .NET" (Microsoft Press, 2002). Microsoft named him a Software Legend in 2002. He wonders whether he should tape down two of his daughter's fingers so she learns how to count in octal. You can contact him at rollthunder.com.*

ReportPlus™
POWERED BY INFRAGISTICS

Unlimited Embedded Dashboards Any Business Can Afford

ReportPlus Embedded

Accelerate Your ROI
We deliver superior ROI compared with other embedded analytics providers. **Our predictable and affordable pricing** package enables you to include data visualizations in an unlimited number of desktop, web or mobile applications.

A Defined Service Engagement
We deliver a built-in partnership by including a robust service engagement and product support as part of our embedded offering at no extra cost.

Tailored Dashboards on Demand
Stay ahead of the competition, continue to innovate, and evolve your app with 30+ beautiful data visualizations you can quickly customize.

INFRAGISTICS

Schedule a custom demo or view pricing at: Infragistics.com/Embedded

To speak with our sales team or a solutions consultant call 1.800.231.8588

ReportPlus Embedded SDK: Infragistics.com/EmbeddedSDK