THE MICROSOFT JOURNAL FOR DEVELOPERS

OCTOBER 2016 VOL 31 NO 10

msdn magazine

THE MICROSOFT JOURNAL FOR DEVELOPERS

OCTOBER 2016 VOL 31 NO 10

msdn magazine

COLUMNS

Microsoft

TEXT CONTROL

ASP.NET MVC REPORTING

The first cross-browser, true WYSIWYG HTML5-based document editor. The Text Control Reporting Framework combines powerful reporting features with an easy-to-use, MS Word compatible word processor.

Give your users an MS Word compatible editor to create powerful documents and reporting templates anywhere - in any browser. Feature-complete including mail merge, sections, headers and footers, drawings and shapes, tables, barcodes and charts.

Live demo and 30-day trial version download at:
www.textcontrol.com/**reporting**

Visual Studio Partner

HTML 5

Call us: 855-533-TEXT

reportingcloud

CREATE DOCUMENTS IN THE

Web API powered reporting platform to create MS Word compatible reports in the cloud.

Use the Text Control ReportingCloud REST web service to create powerful, MS Word compatible reports with JSON data from all clients such as .NET, Java, JavaScript, Ruby, PHP, Python, Android, iOS and many more. Templates can be created and edited directly in any browser.

Free trial account and wrapper documentation at:
www.reporting.cloud

php JS Software ▪ Training ▪ Consulting TEXT CONTROL

msdn magazine

OCTOBER 2016 VOLUME 31 NUMBER 10

ID STATEMENT MSDN Magazine (ISSN 1528-4859) is published monthly 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 www.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. www.magreprints.com/QuickQuote.asp

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: www.meritdirect.com/1105

Reaching the Staff

Staff may be reached via e-mail, telephone, fax, or mail. A list of editors and contact information is also available online at Redmondmag.com.

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

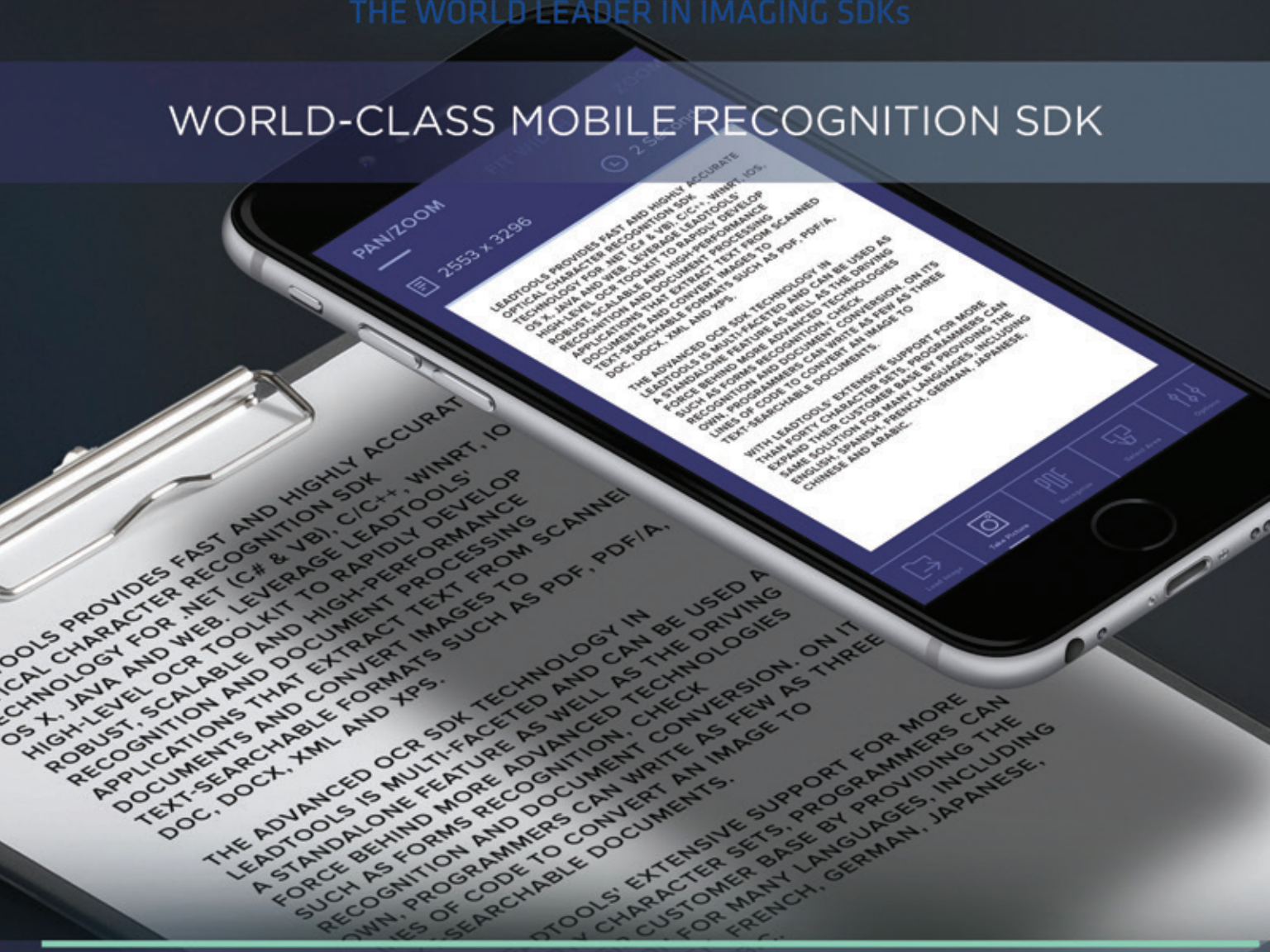
1906 100 2006 AMERICAN BUSINESS MEDIA

WPA Western Publications Association

LEADTOOLS®

THE WORLD LEADER IN IMAGING SDKs

WORLD-CLASS MOBILE RECOGNITION SDK



LEADTOOLS recognition engines enable developers to easily integrate fast and accurate OCR, Barcode, Driver's License, Passport, and Credit Card recognition functionality into their mobile apps.

Download fully functional sample apps built with the LEADTOOLS SDK.

Google play | Download on the App Store | Microsoft

DOWNLOAD OUR 60 DAY EVALUATION
WWW.LEADTOOLS.COM

LEAD TECHNOLOGIES

SALES@LEADTOOLS.COM
800.637.1840

MICHAEL DESMOND

Everything I Need to Know I Learned in 'Monty Python and the Holy Grail'

"Monty Python and the Holy Grail" might be the funniest movie ever made. Filmed on a shoestring budget by the irreverent British comedy troupe Monty Python, the movie tackles the legend of King Arthur and lampoons everything from arranged marriages to medieval superstitions to the French. Given its medieval setting and farcical treatment, you'd think there would be little software developers could learn from the film. As it turns out, "The Holy Grail" offers more than a few lessons for those involved with code.

Be pragmatic: Everyone told the King of Swamp Castle that he was "daft to build a castle on a swamp," and sure enough, his first attempt sank into the muck. "So I built a second one," he says. "That sank into the swamp. So I built a third. That burned down, fell over, then sank into the swamp. But the fourth one stayed up."

This is the kind of thinking that produces epic SAP implementation failures. The King and his melancholy son Alice ... I mean, Herbert … could've spared themselves a lot of trouble if they had listened to trusted advisors and built on another site. Be pragmatic and be willing to shift your plan as events and conditions warrant. Oh, and no singing.

Assume nothing: That simple code update project you were assigned? It could be a killer rabbit. Don't make assumptions. Assess for complexity and scope, watch out for sharp, pointy teeth and budget accordingly.

Commit to process: It's not enough to have a great idea, you have to execute. Sir Bedevere's giant Trojan rabbit ploy worked flawlessly—except he forgot to put anyone in the construct before the French guards rolled it into their castle. A more rigorous process would have ensured the project was complete *before* he deployed it, allowing the knights to take the French "not only by surprise, but totally unarmed!"

Harden your code: At the Bridge of Death, each knight had to correctly answer three questions or face being launched into the chasm. When the Bridgekeeper asked King Arthur the air speed velocity of a laden swallow, Arthur replied, "What do you mean, African or European swallow?" Surprised, the Bridgekeeper muttered, "I don't know that," and was himself launched into the abyss. The unanticipated input crashed the authentication routine and left the bridge unguarded. This is stuff that can be addressed by fuzz tests and error checking.

Value documentation: The knights' quest would never have even reached the Bridge, had Joseph of Arimethea not documented his work in the living rock of the Caves of Caerbannog. And the knights never would have gotten past the killer rabbit without the Book of Armaments and its detailed instructions on using the Holy Hand Grenade of Antioch. Document, document, document!

That simple code update project you were assigned? It could be a killer rabbit.

Consider open source: True fact: The budget for "The Holy Grail" was so small that the crew couldn't afford horses. So cast members took two halves of coconut and clapped them together as they skipped along. Sometimes, free tools are the best tools.

Know when to quit: The Black Knight always triumphs, except when he doesn't. Don't be stubborn—no one is of any use with all their limbs lopped off. Take a lesson from King Arthur and his men, who throughout the movie would cry, "Run away! Run away!" as they retreated from one peril after the next. It looked cowardly and silly, but those retreats allowed the knights to regroup and press on with their quest.

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

Switch to Amyuni PDF

AMYUNI

Create and Edit PDFs in .NET, COM/ActiveX, WinRT & UWP

NEW v5.5

- Edit, process and print PDF 1.7 documents
- Create, fill-out and annotate PDF forms
- Fast and lightweight 32- and 64-bit components for .NET and ActiveX/COM
- New Universal Apps DLLs enable publishing C#, C++, CX or Javascript apps to windows Store
- Updated Postscript/EPS to PDF conversion module

Complete Suite of Accurate PDF Components

- All your PDF processing, conversion and editing in a single package
- Combines Amyuni PDF Converter and PDF Creator for easy licensing, integration and deployment.
- Includes our Microsoft WHQL certified PDF Converter printer driver
- Export PDF documents into other formats such as Jpeg, PNG, XAML or HTML5
- Import and Export XPS files using any programming environment

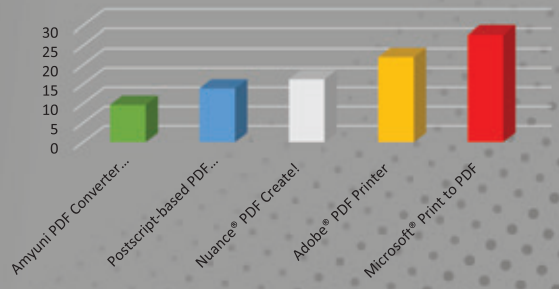
Other Developer Components from Amyuni®

- WebkitPDF: Direct conversion of HTML files into PDF and XAML without the use of a web browser or a printer driver
- PDF2HTML5: Conversion of PDF to HTML5 including dynamic forms
- Postscript to PDF Library: For document workflow applications that require processing of Postscript documents
- OCR Module: Free add-on to PDF Creator uses the Tesseract engine for character recognition
- Javascript engine: Integrate a full Javascript interpreter into your applications to process PDF files or for any other need

High Performance PDF Printer for Desktops and Servers

- Print to PDF in a fraction of the time needed with other tools. WHQL tested for all Windows platforms. Version 5.5 updated for Windows 10 support

Benchmark Testing - Amyuni vs Others
Seconds required to convert a document to PDF



CERTIFIED FOR Windows Server 2008

Windows Server 2012 Certified

Windows 7

Windows

AMYUNI Technologies

USA and Canada
Toll Free: 1866 926 9864
Support: 514 868 9227
sales@amyuni.com

Europe
UK: 0800-015-4682
Germany: 0800-183-0923
France: 0800-911-248

All development tools available at

www.amyuni.com

DINO ESPOSITO

Event-Command-Saga Approach for Business Logic

If we could work out of relatively frozen requirements, any design effort done in advance of development would certainly pay off. If you remember the term "Big Design Up Front," you know what I mean. If not, you can check it out at bit.ly/1jVQr3g. A comprehensive domain model that handles both numerous and intricate workflows and rich and denormalized data views requires a solid and stable knowledge of the domain, while lengthening development time. Put another way, given the relevance that software has today for everyday business, a comprehensive domain model is as bad now as Big Design Up Front was perceived before the Agile movement came along.

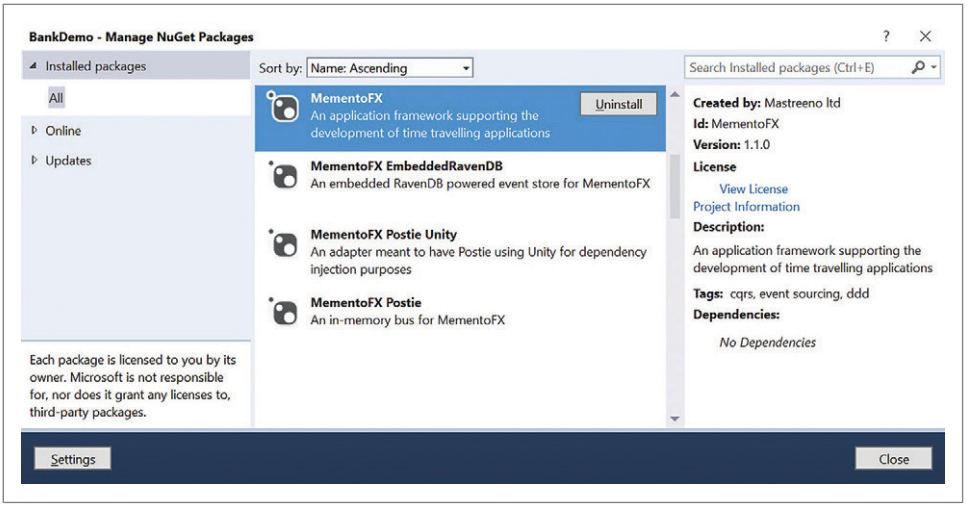


Figure 1 **Installing the MementoFX NuGet Packages**

The Event-Command-Saga (ECS) approach promotes a much more agile way to implement business workflows. It still requires solid knowledge of the processes and business rules, but it doesn't require a big design to be in place before coding starts. Furthermore, it's flexible in accommodating changes and, more important, business points that were initially missed or overlooked.

The ECS approach promotes a message-based formulation of business processes, which is surprisingly closer to the abstraction level of flowcharts. Therefore, it's something that stakeholders can easily communicate and validate. A message-based formulation of business processes is also far easier to understand for developers, even when their understanding of the specific business domain is limited. The term ECS will probably sound new, but the concepts it builds on are the same you might find referred to as CQRS/ES in other sources .

In this month's column, I'll present a .NET-based framework specifically devised to implement the business logic of applications using relatively new concepts such as commands and sagas. For an introduction to the topic, you might want to check out my September 2016 column (msdn.com/magazine/mt767692). The term "business logic" here encompasses both the application and domain logic. The "application logic" is where you implement all use cases that depend on a given front end. The "domain logic," instead, is invariant to use cases and entirely reusable across all flavors of presentation and application layers you might have.

Code download available at msdn.com/magazine/1016magcode.

MementoFX in Action

Let's start with a new ASP.NET MVC project already configured to use common things such as Bootstrap, jQuery, Entity Framework and ASP.NET SignalR. Now add a controller class with a method and related view that displays to users an HTML form. When the user submits the form, the following code is expected to run:

```
[HttpPost]
public ActionResult Apply(NewAccountRequestViewModel input)
{
  _service.ApplyRequestForNewBankAccount(input);
  return RedirectToAction("index", "home");
}
```

At first sight, this is standard code, but the fun lies just under the surface. So open the code for the ApplyRequestForNewBankAccount method in the application layer.

Business-wise, the application user (likely, a bank employee) has just filled out the form through a customer request to open a new account. There's a specific process to start whenever such a new request comes in. You can procedurally code all the steps of the workflow right in the application layer, or you can try the ECS approach. In the latter case, here's what you come up with:

```
public void ApplyRequestForNewBankAccount(NewAccountRequestViewModel input)
{
  var command = new RequestNewBankAccountCommand(
    input.FullName, input.Age, input.IsNew);
  Bus.Send(command);
}
```

The RequestNewBankAccountCommand class is a little bit more than just a Plain Old CLR Object (POCO) class. It's a POCO class but it inherits from Command. In turn, the Command class is defined in one of the NuGet packages that form the MementoFX framework. You then add the packages as shown in **Figure 1**.

The MementoFX framework is made up of three main parts: a core library, an event store and a bus. In the sample configuration, I used the embedded version of RavenDB for storing domain events and an in-memory bus (Postie) for the application layer and sagas to publish and subscribe to events. If you explore the NuGet platform further, you also find a bus component based on Rebus and an event store component based on MongoDB. So now, the following code compiles nicely:

```
public class RequestNewBankAccountCommand : Command
{
  ...
}
```

In the current version of MementoFX, the base Command class is a mere marker and contains no functional code, but this will likely change in future versions. The command wraps up the input parameters for the first step of the business process. To trigger the business process, the command is placed to the bus.

Configuring the MementoFX Environment

The initial setup of MementoFX is easier if you make use of an Inversion of Control (IoC) framework such as Unity. To configure MementoFX, you need to do the following three things: First, initialize the event store of choice. Second, tell MementoFX how to resolve generic interface types to concrete types (this mostly means telling the framework about the type of bus to use, the event store and the event store document). Third, you actually resolve the bus to a concrete instance and bind it to sagas and handlers as appropriate. **Figure 2** summarizes the process.

As shown in **Figure 2**, the bus has two subscribers—the mandatory AccountRequestSaga and the optional AccountRequestDenormalizer. The saga contains the code that does the job of processing the request. Any business logic you might have applies here. The denormalizer will receive information about the aggregate and, if needed, will create a projection of the data just for query purposes.

Figure 2 **Configuring the MementoFX**

```
// Initialize the event store (RAVENDB)
NonAdminHttp.EnsureCanListenToWhenInNonAdminContext(8080);
var documentStore = new EmbeddableDocumentStore
{
  ConnectionStringName = "EventStore",
  UseEmbeddedHttpServer = true
};
documentStore.Configuration.Port = 8080;
documentStore.Initialize();

// Configure the FX
var container = MementoFxStartup
  UnityConfig<InMemoryBus, EmbeddedRavenDbEventStore,
     EmbeddableDocumentStore>(documentStore);

// Save global references to the FX core elements
Bus = container.Resolve<IBus>();
AggregateRepository = container.Resolve<IRepository>();

// Add sagas and handlers to the bus
Bus.RegisterSaga<AccountRequestSaga>();
Bus.RegisterHandler<AccountRequestDenormalizer>();
```

Designing the Saga

A saga is a class that represents a running instance of a business process. Depending on the actual capabilities of the bus you use, the saga can be persisted, suspended and resumed as appropriate. The default bus you have in MementoFX only works in memory. So, any saga is a one-off process that transactionally runs from start to finish.

A saga must have a starter event or command. You indicate the starter message through the interface IAmStartedBy. Any additional message (command or event) the saga knows how to handle is bound through the IHandlesMessage interface:

```
public class AccountRequestSaga : Saga,
  IAmStartedBy<RequestNewBankAccountCommand>,
  IHandleMessages<BankAccountApprovedEvent>
{
  ...
}
```

Both interfaces are made of a single Handle method, as shown here:

```
public void Handle(RequestNewBankAccountCommand message) { ... }
public void Handle(BankAccountApprovedEvent message) { ... }
```

Let's switch back to the HTML form you assumed to have in the UI. When the bank employee clicks to submit the customer's request for a new bank account, a command is pushed to the bus and the bus silently triggers a new saga. Finally, the Handle method of the saga for the specified command is run.

Adding Behavior to the Saga

A saga class is instantiated as shown here:

```
public AccountRequestSaga(
  IBus bus, IEventStore eventStore, IRepository repository)
  : base(bus, eventStore, repository)
{
}
```

It gets a reference to the bus so that the current saga can push new commands and events to the bus for other sagas or handlers and denormalizers to process. This is actually the key factor that enables a flexible and agile design of business workflows. In addition, a saga gets a reference to the repository. In MementoFX, the repository is a façade built on top of the event store. The repository saves and returns aggregates, except that the state of the aggregate is rebuilt every time by replaying all events it went through. Nicely enough, the MementoFX repository also offers an overload to query the state of a given aggregate at a given date.

Here's a saga that would persist the request for a new bank account:

```
public void Handle(RequestNewBankAccountCommand message)
{
  var request = AccountRequest.Factory.NewRequestFrom(
    message.FullName, message.Age, message.IsNew);
  Repository.Save(request);
}
```

In this example, the AccountRequest class is a MementoFX aggregate. A MementoFX aggregate is a plain class derived from a specific parent. Assigning a parent class will save you from the burden of coding a bunch of things as far as management of internal domain events is concerned:

```
public class AccountRequest : Aggregate,
  IApplyEvent<AccountRequestReceivedEvent> { ... }
```

Another interesting aspect of MementoFX aggregates is the IApplyEvent interface. The type associated with the IApplyEvent interface defines a domain event that's relevant for the aggregate to track. Put another way, it means that all events associated with the IApplyEvent interface are saved in the event store for that instance of the aggregate class. Therefore, for a bank account request, you can get to know when it was received, when it was processed, approved, delayed, denied and so forth. Moreover, all events will be stored in their natural order, meaning that it's easy for the framework to take all events up until a given date and return a view of the aggregate at any time in the life of the system. Note that in MementoFX the use of the IApplyEvent is optional, in the sense that you're also welcome to manually persist relevant events in the store when some other method of the aggregate is invoked. The use of the interface is a recommended practice that keeps the code more clear and concise.

When defining an aggregate, you must indicate its unique ID. By convention, MementoFX recognizes as the ID a property with the name of the aggregate class, plus "Id." In this case, it would've been AccountRequestId. If you want to use another name (say, RequestId), you use the AggregateId attribute, as shown here:

```
public void ApplyEvent(
  [AggregateId("RequestId")]
  AccountRequestReceivedEvent theEvent)
{ ... }
```

In C# 6, you can also use the nameof operator to avoiding using a plain constant in compiled code. With MementoFX and the ECS approach, you need to modify a bit the persistence logic you might be used to. For example, when the saga is about to log the request for the account, it uses the factory of AccountRequest to get a new instance. Note that in order to avoid compile time errors, the factory class must be defined within the body of the AccountRequest class:

```
public static class Factory
{
  public static AccountRequest NewRequestFrom(string name, int age, bool isNew)
  {
    var received = new AccountRequestReceivedEvent(Guid.NewGuid(), name, age, isNew);

    var request = new AccountRequest();
    request.RaiseEvent(received);
    return request;
  }
}
```

As you can see, the factory doesn't fill out the newly created instance of the aggregate, it just prepares an event and raises it. The RaiseEvent method belongs to the base Aggregate class, has the effect of adding that event to the current instance of the aggregate and calls ApplyEvent. So, in an apparently intricate way, you're at

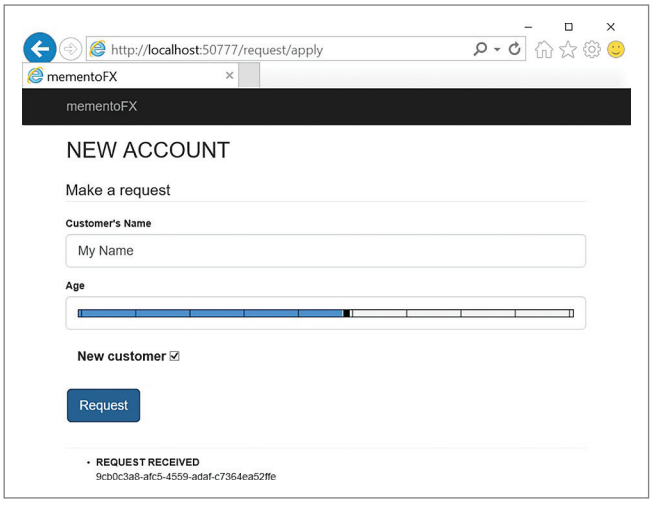


Figure 3 **The Sample MementoFX Application in Action**

the point of returning out of the factory a fully initialized aggregate. The benefit, though, is that not just the aggregate holds its current state, but it also holds all of the relevant events that were brought there in the current operation.

What happens when the saga saves the aggregate to the persistence layer? The Save method of the built-in repository goes through the list of pending events in the aggregate and writes them down to the configured event store. When the GetById method is called instead, it takes the ID to retrieve all the related events and returns an instance of the aggregate that results from replaying all logged events. **Figure 3** shows a UI that's pretty much the one you would imagine out of a standard approach. However, what happens under the hood is quite different. Note that in the UI I used ASP.NET SignalR to bring changes back to the main page.

A Word on Denormalizers

One of the most important changes in software recently is the separation between the model ideal to save data and the model ideal to consume data, as per the CQRS pattern. So far, you just saved an aggregate with all of the information that's relevant to save. Each type of user, though, might have a different set of relevant information for the same aggregate. When this happens, you need to create one or more projections of stored data. A projection in this context is pretty much the same as a view in a SQL Server table. You use denormalizers to create projection of aggregates. A denormalizer is a handler bound to an event pushed to the bus. For example, imagine you need to create a dashboard for the managers responsible for approving new account requests. You might want to offer a slightly different aggregation of the same data perhaps with some indicators that are relevant to the business:

```
public class AccountRequestDenormalizer :
  IHandleMessages<AccountRequestReceivedEvent>
  {
    public void Handle(AccountRequestReceivedEvent message)
    { ... }
}
```

Denormalized data doesn't need to go in the event store. You can reasonably use any database you like for that and most of the time a classic relational engine is the most effective solution.

Wrapping Up

This column offered a glimpse of a new way to organize business logic putting together CQRS and Event Sourcing, but without dealing with the low-level details and intricacies of both patterns. Furthermore, the ECS approach is also close to the real business to favor communication and reduce risk of misunderstanding. MementoFX is on NuGet for you to try out. I can't wait to hear your feedback. ■

Dino Esposito *is the author of "Microsoft .NET: Architecting Applications for the Enterprise" (Microsoft Press, 2014) and "Modern Web Applications with ASP.NET" (Microsoft Press, 2016). A technical evangelist for the .NET and Android platforms at JetBrains, and frequent speaker at industry events worldwide, Esposito shares his vision of software at software2cents@wordpress.com and on Twitter: @despos.*

Thanks to the following Microsoft technical expert for reviewing this article: *Andrea Saltarello*

JULIE LERMAN

Run EF Core on Both .NET Framework and .NET Core

The technology formerly known as Entity Framework 7 (EF7) was renamed to Entity Framework Core (EF Core) in early 2016. EF Core 1.0.0 introduces some great new capabilities, though overall it does have a smaller feature set than EF6. But this doesn't mean EF runs only on .NET Core. You can use EF Core in APIs and applications that require the full .NET Framework, as well as those that target only the cross-platform .NET Core. In this column, I'll walk you through two projects that explore these options. My goal is to alleviate any worries the "Core" moniker might imply: that EF Core only runs on .NET Core. At the same time, I'll explain the steps involved in creating each solution.

EF Core in Full .NET Projects

I'll begin with a project that targets the full .NET Framework. Keep in mind that in Visual Studio 2015, the tooling requires that you have Visual Studio 2015 Update 3, as well as the latest Microsoft ASP.NET and Web Tools. At the time of writing this column (August 2016), the best guide for those installations is the documentation at bit.ly/2bte6Gu.

To keep my data access separate from whatever app will be using it, I'll create it in its own class library. **Figure 1** shows that there's a template that specifically targets a .NET Core class library; but I'm selecting Class Library, the standard option that's always been available for targeting .NET. The resulting project (**Figure 2**) is also "normal"—you can see that there are no project.json files or any .NET Core project assets. Everything looks just the way it always has.

So far, none of this is tied to EF in any way. I could choose EF6 or EF Core at this point, but I'll add EF Core into the project. As always, I can use either the NuGet Package Manager to find and select EF Core or the Package Manager Console window. I'll use the console. Remember that the "entityframework" package is for EF6. To get EF Core, you need to install one of the Microsoft.EntityFrameworkCore packages. I'll use the SqlServer package, which will bring in what EF needs to communicate with SqlServer:

```
install-package Microsoft.EntityFrameworkCore.SqlServer
```

Because that package depends on the main Microsoft.EntityFrameworkCore package, as well as the Microsoft.EntityFrameworkCore.Relational package, NuGet will install those for me at the same time. And because the EF Core package depends on other packages, they'll be installed, too. In all, this process adds the three EF Core packages, as well as 23 others from the newer, more composable .NET on which EF Core relies. Rather than fewer large packages, I get more small packages—but only those my software needs. These will all play well with the standard .NET libraries already in the project.

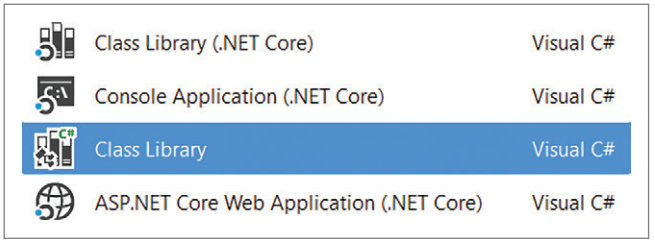


Figure 1 **Creating a Class Library for a Full .NET API**

Next, I'll add in a simple domain class (Samurai.cs) and a DbContext (SamuraiContext.cs) to let EF Core persist my data into a database, as shown in **Figure 3**. EF Core doesn't have the magical connection string inference that EF6 has, so I have to let it know what provider I'm using and what connection string. For simplicity, I'll stick that right in the DbContext's new virtual method: OnConfiguring. I've also created a constructor overload to allow me to pass in the provider and other details as needed. I'll take advantage of this shortly.

Because I'm using the full .NET, which also means I'm targeting full-blown Windows, I have Windows PowerShell available. And this means I get to use the same migrations commands I've always used: add-migration, update-database and so forth. There are some new commands, as well, and you can check out my January 2016 column (msdn.com/magazine/mt614250) to learn all about the EF Core migrations commands. Also, remember I mentioned that packages are smaller and composable? Well, if I want to use migrations, I need to add in the package that contains these commands. As I'm writing this, the tools are still in preview mode so I need to use the -pre parameter. I'll add that package, then I can add a new migration:

```
install-package Microsoft.EntityFrameworkCore.Tools -pre
add-migration init
```

This works as it always has: it creates a new Migrations folder and the migration file, as shown in **Figure 4**. EF Core did change the way it stores model snapshots, which you can read about in the aforementioned January 2016 column.

With the migration in place, the update-database command successfully creates the new EFCoreFullNet database for me in SQL Server localdb.

Finally, I'll add a test project to the solution from the same Unit Test Project template I've always used in Visual Studio. I'll then add a reference to my EFCoreFullNet class library. I don't need my

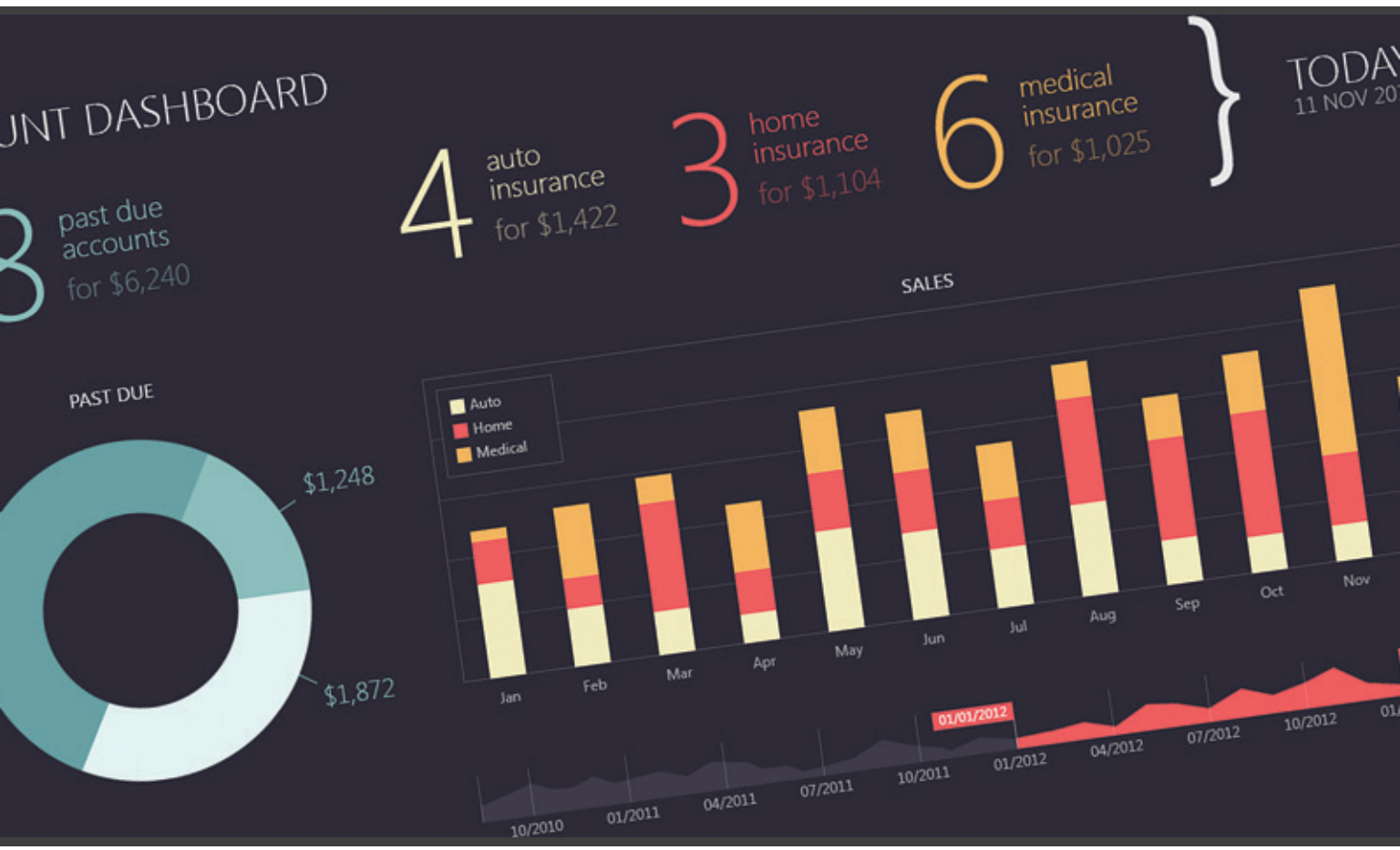
This article relies on preview versions of DotNet CLI and EF tools. All information is subject to change.

Code download available at msdn.com/magazine/1016magcode.

DevExpress®

Dashboards & Analytics

DevExpress Universal ships with everything you'll need to create information-rich decision support systems and to distribute your dashboard solutions royalty-free.



Your Next Great Dashboard Starts Here

Learn how you can create fully customizable Dashboards with our flexible data visualization tools.
devexpress.com/dashboard

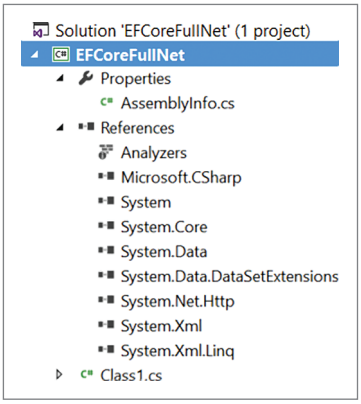


Figure 2 **A Plain Old (and Familiar) .NET Class Library**

test project to use the database to make sure EF Core is working, so rather than installing the SqlServer package, I'll run the following NuGet command against the new test project:

```
install-package microsoft.
  EntityFrameworkCore.InMemory
```

The InMemory provider is a blessing for testing with EF Core. It uses in-memory data to represent the database and the EF cache, and EF Core will interact with the cache in much the same way it works with a database—adding, removing and updating data

Remember that extra constructor I created in the SamuraiContext? The TestEFCoreFullNet tests, shown in **Figure 5**, take advantage of it. Notice that in the constructor of the test class, I created a DbContextOptions builder for the SamuraiContext and then specified it should use the InMemory provider. Then, in the method when I instantiate SamuraiContext, I pass in those options. The SamuraiContext OnConfiguring method is designed to check to see if the options are already configured. If so, it will use them (in this case, the InMemory provider); otherwise, it will move ahead with setting up to work with SqlServer and the connection string I hardcoded into the method.

This test method takes advantage of some specific EF Core features that don't exist in EF6. I wrote about these and other change-tracking features in EF Core in my August 2016 Data Points column (msdn.com/magazine/mt767693). For example, after creating the new samurai object, I add it to the context using the DbContext.Add method, letting EF determine to which DbSet it needs to be tied. Then I save that to the data store, in this case some type of list in memory that the InMemory provider is managing. Next, I modify the samurai object, create a new instance of DbContext and use the new EF Core Update command to make sure SaveChanges will update the stored samurai rather than create a new one. Finally, I query the context for that samurai and use an Assert to ensure that the context does indeed return the updated name.

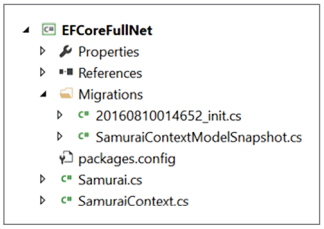


Figure 4 **EF Core Migrations in My Full .NET Class Library**

The particular features I'm using are not the point, however. The point is that I'm doing all of this work with EF Core in a "plain old .NET" project in Windows.

EF Core for CoreCLR: Same Code, Different Dependencies

I could stay in Windows and in Visual Studio 2015 Update 3 to next show you how I can use the same EF Core APIs, the same code and the same tests to target the CoreCLR runtime, but that looks too similar to targeting Windows. Therefore, I'll go to the other extreme and create the CoreCLR variation on my MacBook, explaining the steps as I go through them.

.NET Core doesn't rely on Windows or its tooling. Besides Visual Studio 2015, I could use … well, I suppose Emacs was the popular non-Visual Studio editor of old. However, there are some cross-platform IDEs I can pick from, not just for writing the code but also to get features like debugging and Git support. For example, in the August 2016 issue of *MSDN Magazine*, Alessandro Del Sole walked through building an ASP.NET Core Web site using Visual Studio Code (msdn.com/magazine/mt767698). I could see from

Figure 3 **Samurai Class and SamuraiContext DbContext Class**

```
public class Samurai
  {
    public int Id { get; set; }
    public string Name { get; set;}
  }

public class SamuraiContext : DbContext
  {
    public DbSet<Samurai> Samurais { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder) {
      {
        if (optionsBuilder.IsConfigured == false) {
          optionsBuilder.UseSqlServer(
        @"Data Source=(localdb)\\mssqllocaldb;Initial Catalog=EFCoreFullNet;
                      Integrated Security=True;");
        }

        base.OnConfiguring(optionsBuilder);
      }
    }

    public SamuraiContext(DbContextOptions<SamuraiContext> options)
      : base(options) { }
  }
```

Figure 5 **Testing with EFCore**

```
using EFCoreFullNet;
using Microsoft.EntityFrameworkCore;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System.Linq;

namespace Tests
{
  [TestClass]
  public class TestEFCoreFullNet
  {
    private DbContextOptions<SamuraiContext> _options;

    public TestEFCoreFullNet() {
      var optionsBuilder = new DbContextOptionsBuilder<SamuraiContext>();
      optionsBuilder.UseInMemoryDatabase();
      _options = optionsBuilder.Options;
    }

    [TestMethod]
    public void CanAddAndUpdateSomeData() {
      var samurai = new Samurai { Name = "Julie" };
      using (var context = new SamuraiContext(_options)) {
        context.Add(samurai);
        context.SaveChanges();
      }
      samurai.Name += "San";
      using (var context = new SamuraiContext(_options)) {
        context.Samurais.Update(samurai);
        context.SaveChanges();
      }
      using (var context = new SamuraiContext(_options)) {
        Assert.AreEqual("JulieSan", context.Samurais.FirstOrDefault().Name);
      }
    }
  }
}
```

Figure 6 **Creating a New CoreCLR Library with the dotnet Command**

his screenshots that he was in Windows, but otherwise, the experience is essentially the same on a Mac.

Another cross-platform option is Rider from JetBrains. Rider is designed specifically for C# and the best way to describe it is "ReSharper in its own IDE."

I've already been using Visual Studio Code in Windows and OS X (not just for C#, but also for Node.js) and that's what I'll use to show you EF Core in an app built to target CoreCLR. In fact, because I'm building this solution in OS X, targeting CoreCLR is my only option. The array of available APIs for my library is more limited. However, EF Core is the same set of APIs as when I used it in the full .NET library in the first project.

As you'll see, most of the effort will be in setting up the projects and the dependencies that are specific to targeting CoreCLR. But I can use the same SamuraiContext class to define my EF Core data model and the same CanAddAndUpdateSomeData test method from the previous project to do the same work. The code is the same even though I'm now targeting the more limited runtime and working in an environment that can't use anything *but* .NET Core.

Creating a Library Similar to the .NET Class Library

I've created a folder to contain both my Library and the Test projects, with subfolders for each project. Inside the Library subfolder, I can call dotnet new to create the Library project. **Figure 6** shows that command, along with a confirmation that the project was created. Listing the contents of the folder shows that only two files were created—most important, project.json, which contains the list of required NuGet packages and other relevant project details. The Library.cs file is just an empty class file that I'll delete.

Next, I'll open this new library project in Visual Studio Code. I can just type "code" at the prompt. Visual Studio Code opens with this as the target folder, automatically recognizes the packages listed in the json file and offers to run dotnet restore to fix up the unresolved dependencies. I happily accept the offer.

The project.json file looks like the code in **Figure 7**.

Figure 7 **The Project.json File**

```
{
  "version": "1.0.0-*",
  "buildOptions": {
    "debugType": "portable"
  },
  "dependencies": {},
  "frameworks": {
    "netstandard1.6": {
      "dependencies": {
        "NETStandard.Library": "1.6.0"
      }
    }
  }
}
```

Pretty simple. Libraries don't need all of the ASP.NET Core stuff I'm used to using, just the NETStandard Library. The .NET Standard Library encapsulates what's common across the various places .NET can now run. From the .NET Standard documentation (bit.ly/2b1JoHJ), "The .NET Standard Library is a formal specification of .NET APIs that are intended to be available on all .NET runtimes." So this library I'm building can be used with .NET Core and ASP.NET Core and even .NET applications starting with .NET 4.5. You can see a compatibility grid on the documentation page.

My next step is to add EF Core to the project. Keep in mind that because I'm on a Mac, SqlServer isn't an option. I'll use the PostgreSQL provider for EF Core instead, which goes in the currently empty dependencies section of project.json:

```
"dependencies": {
    "Npgsql.EntityFrameworkCore.PostgreSQL": "1.0.0-*"
  },
  "tools": {
    "Microsoft.EntityFrameworkCore.Tools": "1.0.0-preview2-final"
  },
```

As before, I plan to do migrations. Normally, I'd also add a dependency to the Microsoft.EntityFrameworkCore.Tools package that contains the commands, just as I did for the full .NET version of this library. But because of a current limitation with the Preview 2 tooling, I'll postpone this until a later step in the process. However, I do need to be able to access the commands from this library's folder, so I add the package into a special "tools" section of project.json, as the preceding code shows.

Restoring the packages pulls in not only these two packages, but their dependencies, as well. If you check in the project.lock.json file that's created, you can see all of the packages, including Microsoft.EntityFrameworkCore and Microsoft.EntityFrameworkCore.Relational—the same packages you saw added into the earlier .NET solution.

Now I'll just copy in my Samurai.cs and SamuraiContext.cs files. I have to change the OnConfiguring class to use PostgreSQL and its connection string instead of SQL Server. This is what that bit of code now looks like:

```
optionsBuilder.UseNpgsql(
  "User ID=julie;Password=12345;Host=localhost;Port=5432;Database=EFCoreCoreCLR;
    Pooling=true;");
```

It should be time to run the migrations, but here you'll run into a known limitation of the current Preview2 version of the EFCore tools outside of Visual Studio, which is that an executable project is required to find critical assets. So, again, it's a bit of a pain on first encounter, but not too much in the way of extra effort. Read more about that at bit.ly/2btm40W.

Creating the Test Project

I'll go ahead and add in my test project, which I can then use as my executable project for the migrations. Back at the command line, I go to the Oct2016DataPointsMac/Test subfolder I created earlier and run:

```
dotnet new -t xunittest
```

In Visual Studio Code, you'll see the new project.json listed in the Test folder. Because this project will be responsible for making sure the

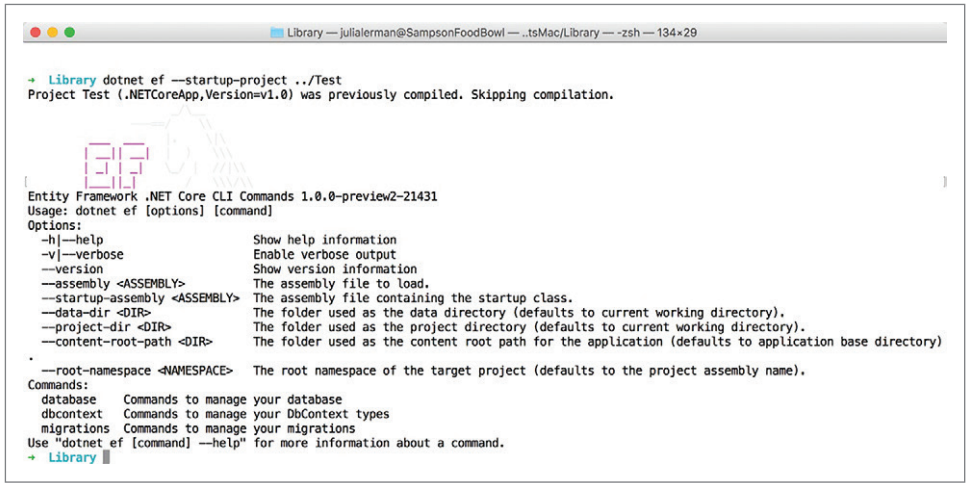


Figure 8 **Leveraging an Executable Project to Enable a Library to Use the EF Migrations Commands**

EF command lines can run, you have to add a reference to the EF Core Tools packages into the dependencies. Additionally, the test project needs a reference to the Library, so I've also added that into the project.json dependencies. Here's the dependencies section after these additions:

```
"dependencies": {
    "System.Runtime.Serialization.Primitives": "4.1.1",
    "xunit": "2.1.0",
    "dotnet-test-xunit": "1.0.0-rc2-192208-24",
    "Library": "1.0.0",
    "Microsoft.EntityFrameworkCore.Tools": "1.0.0-preview2-final"
  },
```

Now I can access the EF Core commands from my Library folder. Notice that in **Figure 8** the command points to the project in the Test folder with the --startup-project parameter. I'll use that with each of the migrations commands.

Running Migrations against the EF Model from the .NET Library

Remember, as I laid out in my column on EFCore migrations, the dotnet ef migrations commands look different from the PowerShell commands, but they lead to the same logic in the migrations API.

First I'll create the migration with:

```
dotnet ef --startup-project ../Test migrations add init
```

This gives the same result as in the .NET solution: a new Migrations folder with the migration and the migration snapshot added.

Now I can create the database with:

```
dotnet ef --startup-project ../Test database update
```

I then verified that the PostgreSQL database, tables and relationships got created. There are a number of tools you can use in OS X to do this. On my Mac, I use the JetBrains cross-platform DataGrip as my database IDE.

Running the Tests in CoreCLR

Finally, I copy the TestEFCoreFullNet class from the earlier solution into my Test folder. Again, I have to make infrastructure changes to use xUnit instead of MS Test: a few namespace changes, removing the TestClass attribute, replacing TestMethod attributes with Fact and replacing Assert.AreEqual with Assert.Equal. Oh and, of course, I rename the class to TestEFCoreCoreClr.

Project.json also needs to know about the InMemory provider, so I add:

```
"Microsoft.EntityFrameworkCore.InMemory": "1.0.0"
```

to the dependencies section, as well, then run "dotnet restore" yet again.

My xUnit test project uses the xUnit command-line test runner. So I'm back to my terminal window to run the tests with the command dotnet test. **Figure 9** shows the output of running the test, which passed with flying colors—except the command-line test runner doesn't provide the satisfying green output for passing tests.

.NET or CoreCLR: Same APIs, Same Code

So now you can see that the code and assemblies related to EF Core are the same whether you target your software to run solely on Windows with the full .NET Framework at your disposal or on CoreCLR on any of the supported environments (Linux, OS X, Windows). I could've done both demonstrations in Visual Studio 2015 on my Windows machine. But I find that focusing the CoreCLR work in an environment that's completely unavailable to the full .NET Framework is an eye-opening way of demonstrating that the EF APIs and my EF-related code are one and the same in both places. The big differences, and all of the extra work, are only related to the target platforms (.NET vs CoreCLR). You can watch me creating a full ASP.NET Core Web API using EF Core 1.0.0 on my MacBook in the video, "First Look at EF Core 1.0" (bit.ly/2cmblqE). For an abbreviated and entertaining demo of the same, check out the video of my DotNetFringe session at bit.ly/2ci7q0T. ■

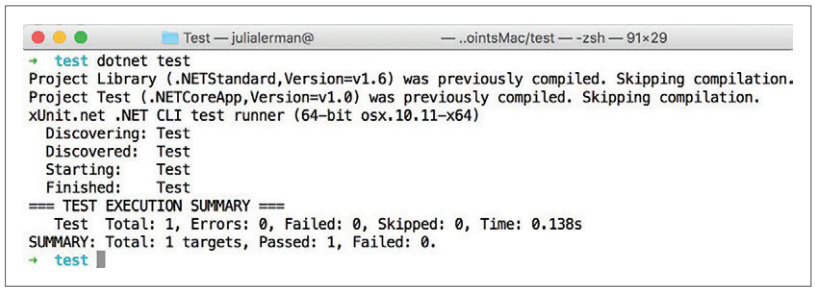


Figure 9 **xUnit Test Output of the Passing Test**

Julie Lerman *is a Microsoft MVP, .NET mentor and consultant who lives in the hills of Vermont. You can find her presenting on data access and other .NET topics at user groups and conferences around the world. She blogs at 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 Fritz*

Know Your Customer

Customers share tons of personal data through an increasing number of channels and applications. We supply the full spectrum of data quality solutions to solve your KYC challenges – to collect, verify, enrich and consolidate clean contact data for a true and complete view of the customer. Microsoft®, Oracle®, Pentaho®, Salesforce® and more.

Face and Emotion Recognition in Xamarin.Forms with Microsoft Cognitive Services

Alessandro Del Sole

At the Build 2016 conference, Microsoft announced a first preview of Cognitive Services (microsoft.com/cognitive-services), a rich set of cross-platform, RESTful APIs that you can leverage to create the next generation of apps based on natural user interaction for any platform on any device. Cognitive Services, also known as "Project Oxford," are based on machine learning and perfectly fit into the conversation-as-a-platform philosophy that Microsoft is willing to bring into the apps ecosystem. At a higher level, the Cognitive Services APIs are available through RESTful services and currently offer the following categories of APIs:

- **Vision:** The Vision services offer APIs that allow you to analyze images and videos to identify faces and emotions, and to detect actionable information. This category includes the Computer Vision, Face, Emotion and Video APIs.

This article discusses:

- Understanding Microsoft Cognitive Services and how to subscribe to the API
- Using Face and Emotion APIs to retrieve face attributes and emotions on pictures
- Manipulating and displaying detection result in Xamarin.Forms
- Leveraging plug-ins for Xamarin to capture and load images

Technologies discussed:

Xamarin.Forms, Microsoft Cognitive Services

Code download available at:

bit.ly/2bgPTm4

- **Speech:** The Speech services offer APIs that make it easier to implement text to speech, natural speech recognition, and even to recognize who's talking with the speaker recognition service. They include the Bing Speech, Custom Recognition Intelligent Service and Speaker Recognition APIs.
- **Language:** The Language services are oriented to natural language understanding, which means detecting and fixing spelling errors, understanding voice commands, and analyzing complex text including sentiments and key phrases. They include the Bing Spell Check, Language Understanding Intelligent Service, Linguistic Analysis, Text Analytics and Web Language Model APIs.
- **Knowledge:** The Knowledge services help applications extend customers' knowledge by finding personalized product recommendations, events, locations, and academic papers or journals. They include the Academic Knowledge, Entity Linking Intelligence Service, Knowledge Exploration Service and Recommendations APIs.
- **Search:** The Search services are based on Bing and allow you to implement powerful search tools in their apps. The included services' names are really self-explanatory: Bing Autosuggest, Bing Image Search, Bing News Search, Bing Video Search and Bing Web Search APIs.

In this article I'll explain how to combine the Face and Emotion APIs to retrieve face details and emotions from pictures you can take from a camera or from an album on disk in a Xamarin.Forms app created with C# and Visual Studio 2015 running on Android, iOS or Windows 10. **Figure 1** shows the results of the article's tutorial. It is important mentioning that, while using Xamarin.Forms for this article, the same

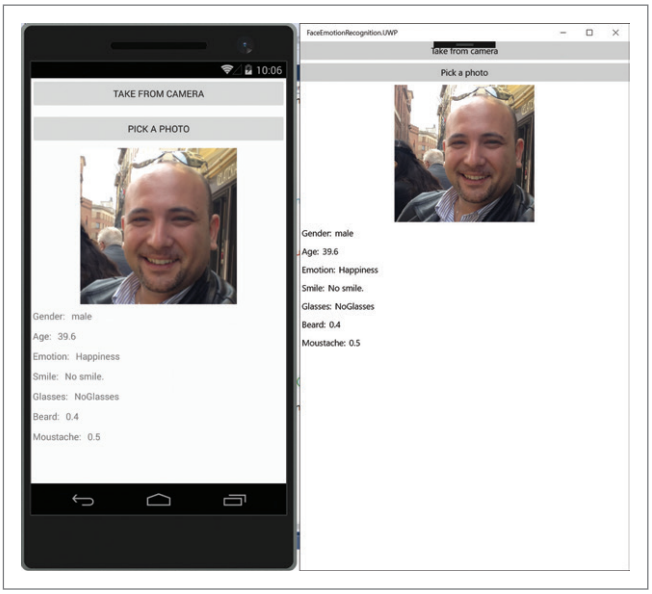


Figure 1 **Face and Emotion Recognition on a Cross-Platform App with Xamarin.Forms (Android Device on Left, Windows 10 Desktop on Right)**

can be done with traditional Xamarin apps, as well as with any other platform that supports REST. I'm assuming you have basic knowledge of creating a Xamarin.Forms app and of the concepts about code sharing; if not, make sure you read my previous articles: "Build a Cross-Platform UX with Xamarin.Forms" (msdn.com/magazine/mt595754) and "Share UI Code Across Mobile Platforms with Xamarin.Forms" (msdn.com/magazine/dn904669).

Subscribing for Cognitive Services APIs

In order to build apps that take advantage of Cognitive Services, you must subscribe to the service in which you're interested. At the moment, Microsoft is offering free trials that you can activate in the subscriptions page (bit.ly/2b2rKD0), but the current plans may be subject to changes in the future. When on the page, register with a Microsoft account, then click "Request new trials." You'll then see a list of available services; make sure you select free previews of both the Face and Emotion APIs. At this point, your subscriptions page will show the list of active services; you should see the Face and Emotion APIs subscriptions. **Figure 2** shows an example based on my subscriptions. Notice how, for each active service, there are two secret keys. You'll need one to invoke the APIs. For now, keep them hidden. You'll unhide the key when creating the Xamarin.Forms app.

> In order to build apps that take advantage of Cognitive Services, you must subscribe to the service in which you're interested.

Generally speaking, Cognitive Services provide RESTful APIs, which means you can interact with these services via HTTP requests on any platform and with any language supporting REST. For example, the following HTTP POST request demonstrates how to send an image to the emotion recognition service for emotion detection:

```
POST https://api.projectoxford.ai/emotion/v1.0/recognize HTTP/1.1
Content-Type: application/json
Host: api.projectoxford.ai
Content-Length: 107
Ocp-Apim-Subscription-Key: YOUR-KEY-GOES-HERE

{ "url": "http://www.samplewebsite.com/sampleimage.jpg" }
```

Of course, you must replace the Ocp-Apim-Subscription-Key with one of your own keys and the fake image URL with a real image address. In exchange, the Emotion recognition service will send back the result of detection as a JSON response, as shown in **Figure 3**.

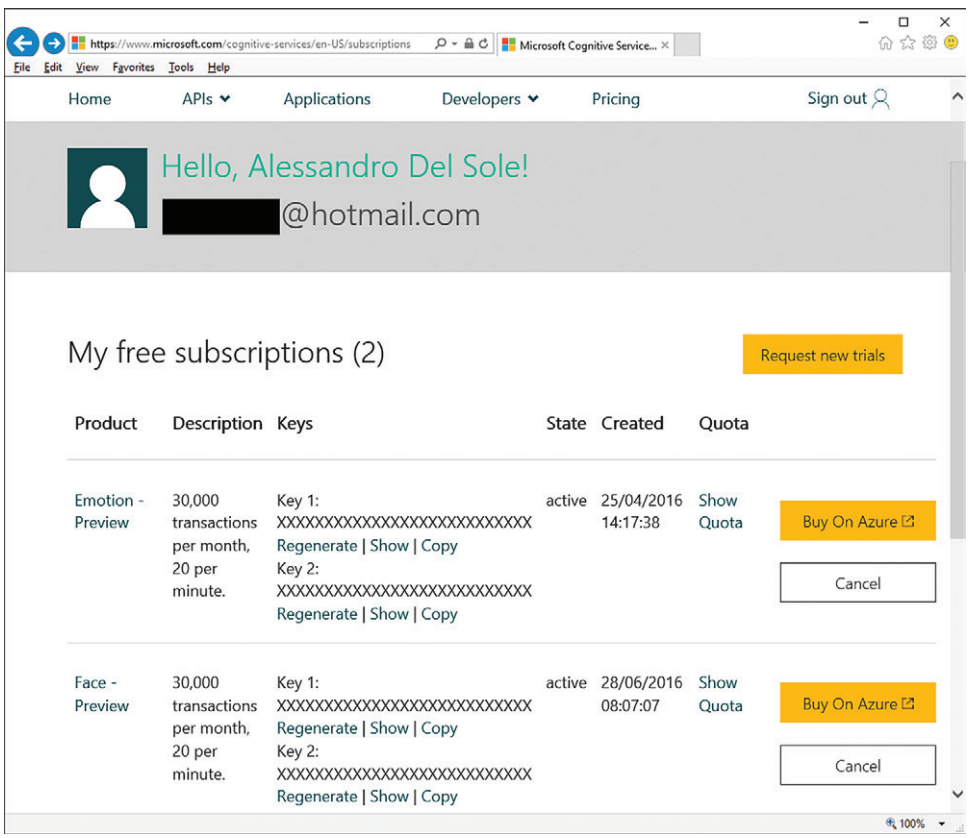


Figure 2 **Activating Subscriptions for Face and Emotion APIs**

Figure 3 **The Emotion Recognition Service Detection Response**

```
[
  {
    "faceRectangle": {
      "height": 70,
      "left": 26,
      "top": 35,
      "width": 70
    },
    "scores": {
      "anger": 2.012591E-11,
      "contempt": 1.95578984E-10,
      "disgust": 1.02281912E-10,
      "fear": 1.16242682E-13,
      "happiness": 1.0,
      "neutral": 9.79047E-09,
      "sadness": 2.91102975E-10,
      "surprise": 1.71011272E-09
    }
  }
]
```

The sample response in Figure 3 shows how the Emotion service returned the rectangle in which a face was detected and an array called scores containing a list of emotions and a value between 0 and 1 that indicates how likely the emotion is to be true. In general, sending HTTP requests to RESTful services and expecting a JSON response is a common approach with all of the Cognitive Services. However, for .NET developers working with C#, Microsoft is also offering client portable libraries that you can download from NuGet and that make it easier to interact with services in managed code and in a fully object-oriented way. This is the case of the Face and Emotion APIs, as you'll see shortly. Don't forget to check out the official documentation, which contains examples based on both the REST approach and on client libraries where available (bit.ly/2b2KJrB). Now that you've registered for both services and you have your keys, it's time to create a cross-platform app with Xamarin.Forms and Microsoft Visual Studio 2015.

Creating a Xamarin.Forms Application

As you know, you can create a cross-platform app with Xamarin.Forms by choosing either the Portable or the Shared project template. Because I'll explain how to leverage client libraries for the Cognitive Services APIs, the sample application is based on the Portable Class Library (PCL) model. In Visual Studio 2015, select File | New Project. If you've installed the latest updates from Xamarin (xamarin.com/download), you'll find a new project template called Blank Xaml App (Xamarin.Forms Portable) under the Visual C#, Cross-Platform node of the New Project dialog. This is an interesting template that provides a blank XAML page, and avoids the need to create one manually. **Figure 4** shows the new template.

> It's your responsibility, as the developer, to check for network availability.

Call the solution FaceEmotionRecognition and click OK. During the generation of the solution, you'll be asked to specify the minimum target version for the Universal Windows Platform (UWP) project. This is left to your choice, but I recommend targeting the highest version available.

Introducing Plug-ins for Xamarin

The sample application will use the Cognitive Services APIs to recognize face details and emotions from pictures, using existing pictures from the device or taking new pictures from the camera. This implies that the app will need to access the Internet to connect to the services and will need to provide the ability of taking and selecting pictures. While an app can easily connect to a network, it's your responsibility, as the developer, to check for network availability. Actually, features like checking for network availability and taking pictures would require writing specific code in the Android, iOS and Windows projects. Luckily enough, Xamarin supports plug-ins that you can use in Xamarin.Forms and that you can install to the PCL project, so that they'll do the job for you. A plug-in is a library installed from NuGet that wraps the native APIs into a common code implementation and invoked in the PCL project. There's a large number of plug-ins—some developed and supported by Xamarin and others created and published by the developer community. Plug-ins are all open source and listed

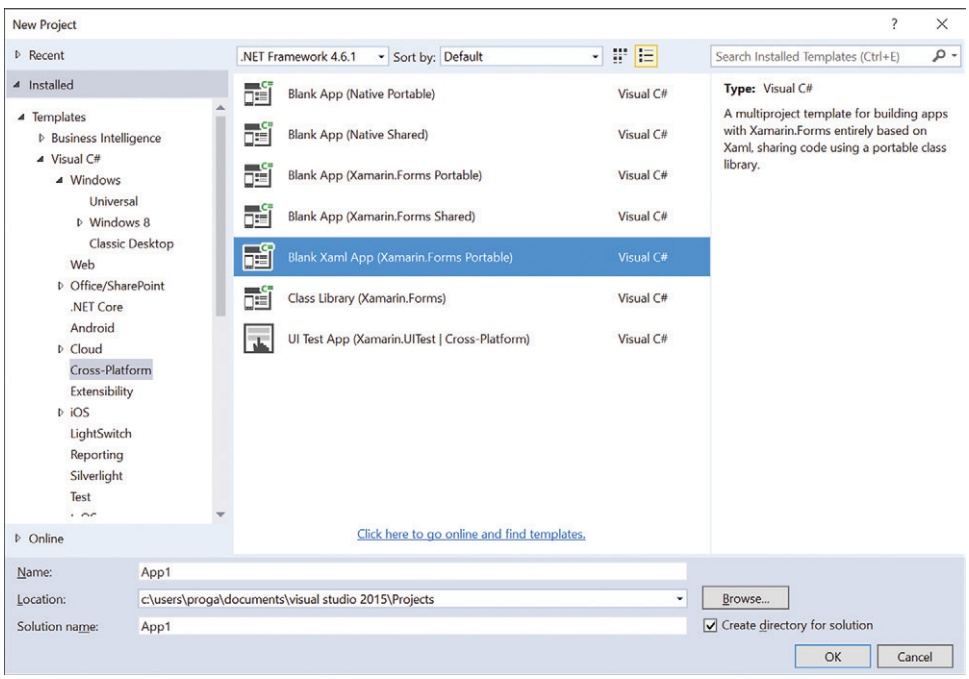


Figure 4 **Creating a New Xamarin.Forms Application**

on GitHub at bit.ly/29XZ3VM. In this article I'll show how to use the Connectivity and Media plug-ins.

Installing NuGet Packages

When the solution is ready, the first thing you need to do is install the following NuGet packages:

- **Microsoft.ProjectOxford.Face:** Installs the client library for the Face APIs and must be installed to the PCL project only.
- **Microsoft.ProjectOxford.Emotion:** Installs the client library for the Emotion APIs and, like for the Face API, must be installed to the PCL project only.
- **Xam.Plugin.Connectivity:** Contains the Connectivity plug-in for Xamarin.Forms and must be installed to all the projects in the solution.
- **Xam.Plugin.Media:** Contains the Media plug-in for Xamarin.Forms and, like the Connectivity API, must be installed to all the projects in the solution.

Once you've installed the required NuGet packages, make sure you build the solution before writing code so that all references will be refreshed.

Figure 5 **The UI for the Main Page**

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  xmlns:local="clr-namespace:FaceEmotionRecognition"
  xmlns:conv="clr-namespace:FaceEmotionRecognition.
    Converters;assembly=FaceEmotionRecognition"
    x:Class="FaceEmotionRecognition.MainPage">

  <StackLayout Orientation="Vertical">
    <Button x:Name="TakePictureButton" Clicked="TakePictureButton_Clicked"
      Text="Take from camera"/>
    <Button x:Name="UploadPictureButton" Clicked="UploadPictureButton_Clicked"
      Text="Pick a photo"/>

    <ActivityIndicator x:Name="Indicator1" IsVisible="False" IsRunning="False" />
    <Image x:Name="Image1" HeightRequest="240" />
    <StackLayout Orientation="Horizontal" Padding="3">
      <Label Text="Gender: "/>
      <Label x:Name="GenderLabel" Text="{Binding Path=Gender}" />
    </StackLayout>
    <StackLayout Orientation="Horizontal" Padding="3">
      <Label Text="Age: "/>
      <Label x:Name="AgeLabel" Text="{Binding Path=Age}"/>
    </StackLayout>
    <StackLayout Orientation="Horizontal" Padding="3">
      <Label Text="Emotion: "/>
      <Label x:Name="EmotionLabel" Text="{Binding Path=Emotion}"/>
    </StackLayout>
    <StackLayout Orientation="Horizontal" Padding="3">
      <Label Text="Smile: "/>
      <Label x:Name="SmileLabel"
        Text="{Binding Path=Smile}"/>
    </StackLayout>
    <StackLayout Orientation="Horizontal" Padding="3">
      <Label Text="Glasses: "/>
      <Label x:Name="GlassesLabel" Text="{Binding Path=Glasses}"/>
    </StackLayout>
    <StackLayout Orientation="Horizontal" Padding="3">
      <Label Text="Beard: "/>
      <Label x:Name="BeardLabel"
        Text="{Binding Path=Beard}"/>
    </StackLayout>
    <StackLayout Orientation="Horizontal" Padding="3">
      <Label Text="Moustache: "/>
      <Label x:Name="MoustacheLabel"
        Text="{Binding Path=Moustache}"/>
    </StackLayout>
  </StackLayout>
</ContentPage>
```

Designing the UI

The sample application's UI consists of a single page. For the sake of simplicity, I'll use the auto-generated MainPage.xaml file. This page defines two buttons, one for taking a picture from the camera and one for uploading an existing image; an ActivityIndicator control that will show a busy status while waiting for a response from the service; an Image control that will display the selected image; a number of labels, within StackLayout panels, that are data-bound to a custom class that will contain the result of detections over the selected picture. **Figure 5** shows the full XAML code for the page.

> Once you've installed the required NuGet packages, make sure you build the solution before writing code, so that all references will be refreshed.

The next step is to prepare a place to store the result of the face and emotion detection.

Storing Detection Results with a Class

Instead of manually populating labels in the UI with the results of face and emotion detection, it's best practice to create a custom class. Not only is this a more object-oriented approach, but it also allows for data binding the class' instance to the UI. That said, let's create a new class called FaceEmotionDetection:

```
public class FaceEmotionDetection
{
  public string Emotion { get; set; }
  public double Smile { get; set; }
  public string Glasses { get; set; }
  public string Gender { get; set; }
  public double Age { get; set; }
  public double Beard { get; set; }
  public double Moustache { get; set; }
}
```

Each property has a self-explanatory name and will store information that comes from the combination of both the Face and Emotion APIs.

Declaring the Service Clients

Before you write any other code, it's a good idea to add the following using directives:

```
Using Microsoft.ProjectOxford.Emotion;
Using Microsoft.ProjectOxford.Emotion.Contract;
Using Microsoft.ProjectOxford.Face;
Using Microsoft.ProjectOxford.Face.Contract;
Using Plugin.Connectivity;
Using Plugin.Media;
```

These will simplify the invocation to object names for both the Cognitive Services APIs and the plug-ins. The Face APIs and the Emotion APIs provide the Microsoft.ProjectOxford.Face.FaceServiceClient and Microsoft.ProjectOxford.Emotion.EmotionServiceClient classes, which connect to the Cognitive Services and respectively return information about face and emotion details.

BEST SELLER

DevExpress DXperience 16.1 | 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

BEST SELLER

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

BEST SELLER

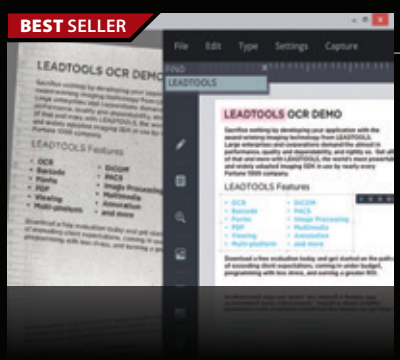
Aspose.Total for .NET | from $2,939.02

ASPOSE
Your File Format APIs

Every Aspose .NET component in one package.

- Programmatically manage popular file formats including Word, Excel, PowerPoint and PDF
- Work with charts, diagrams, images, project plans, emails, barcodes, OCR and OneNote files alongside many more document management features in .NET applications
- Common uses also include mail merging, adding barcodes to documents, building dynamic reports on the fly and extracting text from most document types

BEST SELLER



LEADTOOLS Document Imaging SDKs V19 | from $2,995.00 SRP

LEAD TECHNOLOGIES

Add powerful document imaging functionality to desktop, tablet, mobile & web applications.

- Universal document viewer & conversion framework for PDF, Office, CAD, TIFF & more
- OCR, MICR, OMR, ICR and Forms Recognition supporting structured & unstructured forms
- PDF SDK with text edit, hyperlinks, bookmarks, digital signature, forms, metadata
- Barcode Detect, Read, Write for UPC, EAN, Code 128, Data Matrix, QR Code, PDF417
- Zero-footprint HTML5/JavaScript UI Controls & Web Services

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

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

European Headquarters
ComponentSource
2 New Century Place
East Street
Reading, Berkshire
RG1 4ET
United Kingdom

Asia / Pacific Headquarters
ComponentSource
3F Kojimachi Square Bldg
3-3 Kojimachi Chiyoda-ku
Tokyo
Japan
102-0083

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

VISA DISCOVER

GSA Schedule Contract GS-35F-0188R

Figure 6 **Selecting a Picture from Disk**

```
private async void UploadPictureButton_Clicked(object sender, EventArgs e)
{
  if (!CrossMedia.Current.IsPickPhotoSupported)
  {
    await DisplayAlert("No upload", "Picking a photo is not supported.", "OK");
    return;
  }

  var file = await CrossMedia.Current.PickPhotoAsync();
  if (file == null)
    return;

  this.Indicator1.IsVisible = true;
  this.Indicator1.IsRunning = true;

  Image1.Source = ImageSource.FromStream(() => file.GetStream());

  this.Indicator1.IsRunning = false;
  this.Indicator1.IsVisible = false;
}
```

What you first need to do is declare an instance of both, passing your secret key to the constructor, as shown here:

```
private readonly IFaceServiceClient faceServiceClient;
private readonly EmotionServiceClient emotionServiceClient;

public MainPage()
{
  InitializeComponent();

  // Provides access to the Face APIs
  this.faceServiceClient = new FaceServiceClient("YOUR-KEY-GOES-HERE");
  // Provides access to the Emotion APIs
  this.emotionServiceClient = new EmotionServiceClient("YOUR-KEY-GOES-HERE");
}
```

Notice that you must supply your own secret keys. Both the Face and Emotion API secret keys can be found in the subscriptions page of the Microsoft Cognitive Services portal (bit.ly/2b2rKDO), as shown in **Figure 2**.

Capturing and Loading Images

In Xamarin.Forms, accessing both the camera and the file system would require writing platform-specific code. A simpler approach is using the Media plug-in for Xamarin.Forms, which lets you pick pictures and videos from disk and take pictures and videos with the camera from the PCL project, and with just a few lines of code. This plug-in exposes a class called CrossMedia, which exposes the following members:

- **Current:** Returns a singleton instance of the CrossMedia class.
- **IsPickPhotoSupported** and **IsPickVideoSupported:** Bool properties that return true if the current device supports selecting pictures and videos from disk.
- **PickPhotoAsync** and **PickVideoAsync:** Methods that invoke the platform-specific UI to select a local picture or video, respectively, and return an object of type MediaFile.
- **IsCameraAvailable:** A bool property that returns true if the device has a built-in camera.
- **IsTakePhotoSupported** and **IsTakeVideoSupported:** Bool properties that return true if the current device supports taking pictures and videos from the camera.
- **TakePhotoAsync** and **TakeVideoAsync:** Methods that launch the built-in camera to take a picture or video, respectively, and return an object of type MediaFile.

Do not forget to set the proper permissions in the app manifest to access the camera. For instance, in a UWP project you need both the Webcam and Pictures Library permissions, while on Android you need the CAMERA, READ_EXTERNAL_STORAGE, and WRITE_EXTERNAL_STORAGE permissions. Forgetting to set the required permissions will result in runtime exceptions. Now let's write the Clicked event handler for the UploadPictureButton, which is shown in **Figure 6**.

The code first checks if selecting pictures is supported, showing an error message if IsPickPhotoSupported returns false. PickPhotoAsync (as well as PickVideoAsync) returns an object of type MediaFile, which is a class defined in the Plugin.Media namespace and that represents the selected file. You must invoke its GetStream method to return a stream that can be used as the source for the Image control through its FromStream method. Taking a picture with the camera is also very easy, as shown in **Figure 7**.

The point of interest here is that TakePhotoAsync takes a parameter of type StoreCameraMediaOptions, an object that lets you specify where and how to save a picture. You can set the SaveToAlbum property as true if you want the picture to be saved to the local camera roll, or you can set the Directory property if you want to save to a different folder. As you can see, with very limited effort and with a few lines of code, your app can easily leverage an important capability of all the supported platforms.

Figure 7 **Taking a Picture with the Camera**

```
private async void TakePictureButton_Clicked(object sender, EventArgs e)
{
  await CrossMedia.Current.Initialize();

  if (!CrossMedia.Current.IsCameraAvailable || !CrossMedia.Current.
    IsTakePhotoSupported)
  {
    await DisplayAlert("No Camera", "No camera available.", "OK");
    return;
  }

  var file = await CrossMedia.Current.TakePhotoAsync(new StoreCameraMediaOptions
  {
    SaveToAlbum = true,
    Name = "test.jpg"
  });

  if (file == null)
    return;

  this.Indicator1.IsVisible = true;
  this.Indicator1.IsRunning = true;

  Image1.Source = ImageSource.FromStream(() => file.GetStream());

  this.Indicator1.IsRunning = false;
  this.Indicator1.IsVisible = false;
}
```

Detecting Emotions and Implementing Face Recognition

Now it's time to implement face and emotion recognition. Because this is an introductory article, I'll focus on simplicity. I'll show how to implement detection over one single face in a picture and I'll describe the most important objects and members in the APIs. I'll also give you suggestions on how to implement more detailed detections where appropriate. Based on these assumptions, let's start writing an asynchronous method that performs detections. The first piece is about emotion detection and it looks like this:

```
private async Task<FaceEmotionDetection>
DetectFaceAndEmotionsAsync(MediaFile inputFile)
{
  try
  {
    // Get emotions from the specified stream
    Emotion[] emotionResult = await
      emotionServiceClient.RecognizeAsync(inputFile.GetStream());

    // Assuming the picture has one face, retrieve emotions for the
    // first item in the returned array
    var faceEmotion = emotionResult[0]?.Scores.ToRankedList();
```

The method receives the MediaFile that's produced by selecting or taking a picture. Detecting emotions over faces on a picture is straightforward, because you simply invoke the RecognizeAsync method from the EmotionServiceClient class' instance. This method can receive either a stream or a URL as an argument. In this case, it gets a stream from the MediaFile object. RecognizeAsync returns an array of Emotion objects. Each Emotion in the array stores emotions detected on a single face in a picture. Assuming the selected picture has just one face, the code retrieves the first item in the array. The Emotion type exposes a property called Scores, which contains a list of eight emotion names and their approximate value. More specifically, you get an IEnumerable<string, float>. By invoking its ToRankedList method, you can get a sorted list of detected emotions. The APIs cannot detect a single emotion precisely. Instead, they detect a number of possible emotions. The highest value returned is approximately the actual emotion on the face, but there are still other values that could be checked. The highest value in this list represents the emotion with the highest level of estimated likelihood, which is possibly the actual emotion on a face. For a better understanding, consider the following ranked list of emotions retrieved with the help of the debugger's data tips, which is based on the sample picture shown in **Figure 1**:

```
[0] = {[Happiness, 1]}
[1] = {[Neutral, 1.089301E-09]}
[2] = {[Surprise, 7.085784E-10]}
[3] = {[Sadness, 9.352855E-11]}
[4] = {[Disgust, 4.52789E-11]}
[5] = {[Contempt, 1.431213E-11]}
[6] = {[Anger, 1.25112E-11]}
[7] = {[Fear, 5.629648E-14]}
```

As you can see, Happiness has a value of 1, which is the highest in the list and is the estimated likelihood of the actual emotion. The next step is detecting face attributes. The FaceServiceClient class exposes the DetectAsync method, which is extremely powerful. Not only can it retrieve face attributes such as gender, age, and smile, but it can also recognize people, return the face rectangle (the area on the picture where the face was detected), and 27 face landmark points that let an app identify information such as the position of nose, mouth, ears, and eyes on the picture. DetectAsync has the following signature:

```
Task<Contract.Face[]> DetectAsync(Stream imageStream,
  bool returnFaceId = true, bool returnFaceLandmarks = false,
  IEnumerable<FaceAttributeType> returnFaceAttributes = null);
```

In its most basic invocation, DetectAsync requires a stream pointing to a picture or a URL and returns the face rectangle, while the returnFaceId and returnFaceLandmarks optional parameters, respectively, let you identify a person and return face landmarks. The Face APIs let you create groups of people and assign an id to each person so that you can easily perform recognition. Face landmarks are instead useful to identify a face's characteristics and will be available through the FaceLandmarks property of the Face object. Both identification and landmarks are beyond the scope of this article, but you can find more about these topics at bit.ly/2adPvoP and bit.ly/2ai9WjV, respectively. Similarly, I won't show you how to use face landmarks, but these are stored in the FaceLandmarks property of the Face object. In the current sample scenario, the goal is to retrieve face attributes. The first thing you need is an array of the FaceAttributeType enumeration, which defines the list of attributes you want to retrieve:

```
// Create a list of face attributes that the
// app will need to retrieve
var requiredFaceAttributes = new FaceAttributeType[] {
  FaceAttributeType.Age,
  FaceAttributeType.Gender,
  FaceAttributeType.Smile,
  FaceAttributeType.FacialHair,
  FaceAttributeType.HeadPose,
  FaceAttributeType.Glasses
  };
```

Next, invoke DetectAsync, passing the image stream and the face attributes list. The returnFaceId and returnFaceLandmarks arguments are false because the related information is unnecessary at this point. The method invocation looks like this:

```
// Get a list of faces in a picture
var faces = await faceServiceClient.DetectAsync(inputFile.GetStream(),
  false, false, requiredFaceAttributes);

// Assuming there is only one face, store its attributes
var faceAttributes = faces[0]?.FaceAttributes;
```

> The FaceServiceClient class exposes the DetectAsync method, which is extremely powerful.

DetectAsync returns an array of Face objects, each representing a face in the picture. The code takes the first item in the array, which represents one single face, and retrieves its face attributes. Notice how the last line uses the null conditional operator (?), introduced with C# 6, that returns null if the first element in the array is also null, instead of throwing a NullReferenceException. More about this operator can be found at bit.ly/2bc8VZ3. Now that you have both face and emotion information, you can create an instance of the FaceEmotionDetection class and populate its properties, as demonstrated in the following code:

```
FaceEmotionDetection faceEmotionDetection = new FaceEmotionDetection();
faceEmotionDetection.Age = faceAttributes.Age;
faceEmotionDetection.Emotion = faceEmotion.FirstOrDefault().Key;
faceEmotionDetection.Glasses = faceAttributes.Glasses.ToString();
faceEmotionDetection.Smile = faceAttributes.Smile;
faceEmotionDetection.Gender = faceAttributes.Gender;
faceEmotionDetection.Moustache = faceAttributes.FacialHair.Moustache;
faceEmotionDetection.Beard = faceAttributes.FacialHair.Beard;
```

A few considerations at this point:

- The highest value in the emotions list is taken by invoking FirstOrDefault over the result of the invocation to the Scores.ToRankedList method, which returns an IEnumerable<string, float>.
- The value returned by FirstOrDefault here is an object of type KeyValuePair<string, float> and the Key of type string

BUSINESS FILE APIS

TRY RISK FREE - 30 Day Trial

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

Aspose.Total

Aspose.Cells
XLS, CSV, PDF, SVG, HTML, PNG
BMP, XPS, JPG, SpreadsheetML...

Aspose.Words
DOC, RTF, PDF, HTML, PNG
ePUB, XML, XPS, JPG...

Aspose.Pdf
PDF, XML, XSL-FO, HTML, BMP
JPG, PNG, ePUB...

Aspose.Slides
PPT, POT, ODP, XPS, HTML
PNG, PDF...

Aspose.BarCode
JPG, PNG, BMP, GIF, TIF, WMF
ICON...

Aspose.Tasks
XML, MPP, SVG, PDF, TIFF
PNG...

Aspose.Email
MSG, EML, PST, MHT
OST, OFT...

Aspose.Imaging
PDF, BMP, JPG, GIF, TIFF
PNG...

+ more!

.NET | Java | Cloud

Download FREE trial at www.aspose.com.

Contact Us:
US: +1 903 306 1676
EU: +44 141 628 8900
AU: +61 2 8006 6987
sales@aspose.com

ASPOSE
File Format APIs

stores the emotion name in a human-readable text that will be shown in the UI.

- Glasses is an enumeration that specifies if the detected face is wearing glasses and what kind. The code invokes ToString for the sake of simplicity, but you could definitely implement a converter for different string formatting.

The code invokes ToString for the sake of simplicity, but you could definitely implement a converter for different string formatting.

The final block in the method body returns the instance of the FaceEmotionDetection class and implements exception handling:

```
    return faceEmotionDetection;
  }
  catch (Exception ex)
  {

    await DisplayAlert("Error", ex.Message, "OK");
    return null;
  }
}
```

The last thing you have to do is invoke the custom DetectFaceAndEmotionAsync method. You can do this inside both Clicked event handlers, just before setting to false the IsRunning and IsVisible properties of the ActivityIndicator control:

```
FaceEmotionDetection theData = await DetectFaceAndEmotionsAsync(file);
this.BindingContext = theData;

this.Indicator1.IsRunning = false;
this.Indicator1.IsVisible = false;
```

The BindingContext property of the page receives an instance of the FaceEmotionDetection class as the data source and data-bound children controls will automatically show the related information. With patterns like Model-View-ViewModel, you would wrap the result with a ViewModel class. After a lot of work, you're ready to test the application.

Testing the Application

Select the platform of your choice and press F5. If you use the Microsoft emulators, you can take advantage of the emulator tools to select a physical webcam to take pictures, and you can simulate an SD card to upload files. **Figure 1** shows the result of the detection on a picture of me, on an Android device and on Windows 10 running in desktop mode.

The Face and Emotion APIs did an amazing job because the returned values are very close to the truth, though still approximate. It's worth mentioning that the FaceEmotionDetection class has some properties of type double, such as Smile, Beard and Moustache. They return numeric values, which might not make much sense for the end user in a real-world app. So, in case you want to convert those numeric values into human-readable strings, you might consider implementing value converters and the IValueConverter interface (bit.ly/2bZn01J).

Implementing Network Connectivity Check

A well-designed app that needs to access resources on the Internet should always check for connection availability first. As for accessing the camera and the file system, in Xamarin.Forms checking for connection availability should require platform-specific code. Fortunately, the Connectivity plug-in comes in to help, providing a shared way to perform this check from the PCL project directly. The plug-in offers a class called CrossConnectivity with its Current property that represents a singleton instance of the class. It exposes a bool property called IsConnected that simply returns true if a connection is available. To check for network availability in the sample application, simply place the following code after the declaration of the DetectFaceAndEmotionAsync method:

```
private async Task<FaceEmotionDetection>
  DetectFaceAndEmotionsAsync(MediaFile inputFile)
{
  if(!CrossConnectivity.Current.IsConnected)
  {
    await DisplayAlert("Network error",
      "Please check your network connection and retry.", "OK");
    return null;
  }
```

The class also exposes the following interesting members:

- **ConnectivityChanged:** An event that's raised when the connection state changes. You can subscribe this event and get information on the connectivity status via an object of type ConnectivityChangedEventArgs.
- **BandWidths:** A property that returns a list of available bandwidths for the current platform.

A well-designed app that needs to access resources on the Internet should always check for connection availability first.

Additional information about the Connectivity plug-in can be found at bit.ly/2bbU7wu.

Wrapping Up

Microsoft Cognitive Services provide RESTful services and rich APIs based on machine learning that let you create the next generation of apps. By combining the power of these services with Xamarin, you'll be able to bring natural user interaction to your cross-platform apps for Android, iOS and Windows, offering customers an amazing experience. ■

Alessandro Del Sole *has been a Microsoft MVP since 2008. Awarded MVP of the Year five times, he has authored many books, eBooks, instructional videos and articles about .NET development with Visual Studio. Del Sole works as a solution developer expert for Brain-Sys (brain-sys.it), focusing on .NET development, training and consulting. You can follow him on Twitter: @progalex.*

Thanks to the following technical experts for reviewing this article: *James McCaffrey and James Montemagno*

Working with Memory Limits and Task Priorities in the UWP

Andrew Whitechapel

Far more than any other app platform, the Universal Windows Platform (UWP) supports a vast range of background activities. If these were allowed to compete for resources in an uncontrolled manner, it would degrade the foreground experience to an unacceptable level. All concurrent processes compete for system resources—memory, CPU, GPU, disk and network I/O, and so on. The system Resource Manager encapsulates rules for arbitrating this contention, and the two most important mechanisms are memory limits and task priorities.

The promise of the UWP is that a developer can build an app that will run successfully on a wide range of Windows 10 platforms, from a minimalist IoT device, to the full range of mobile and desktop devices, plus Xbox and HoloLens. Resource policy applies to all Windows 10 platforms, and most policy is common across the range—specifically to support the UWP promise of consistency. That said, some aspects of policy do vary, because different platforms support different sets of hardware devices with different capabilities.

This article discusses:

- How to work with memory limits, and how this is different from previous releases
- How dynamic scenario-based task prioritization affects your app
- The APIs that you can use to get insight into resource policy and app resource usage
- Making the best use of the Visual Studio memory diagnostics tools

Technologies discussed:

Universal Windows Platform, Visual Studio 2015

Code download available at:

msdn.com/magazine/1016magcode

Far more than any other app platform, the Universal Windows Platform (UWP) supports a vast range of background activities.

So, for example, the memory limits on a Lumia 950 phone are almost identical to those on a HoloLens because these two devices have similar RAM characteristics and other hardware capabilities. Conversely, the Lumia 950 limits are significantly higher than on a Lumia 650, which has far less physical RAM and a lower hardware specification, generally. Pagefile is another factor: Desktop devices have a dynamically sizeable pagefile that's also often very fast, whereas on all other Windows 10 devices, the pagefile is small, slow and a fixed-size. This is one reason why memory limits are completely removed on desktop, but enforced on all other devices.

In a few well-defined scenarios, memory limits can also vary at different times on the same device, so apps should take advantage of

the Windows.System.MemoryManager APIs to discover the limit that's actually applied at any point in time. This API will always reliably tell the app its current limit and its current usage—and these same values are exactly the values that the Resource Manager uses in its own internal calculations. In the following example, the app pays attention to its memory limit, and before it attempts a memory-intensive operation, it checks to see that it does in fact have enough headroom available for this operation:

```
private void TryMemoryIntensiveOperation(){
  ulong limit = MemoryManager.AppMemoryUsageLimit;
  ulong usage = MemoryManager.AppMemoryUsage;
  ulong headroom = limit - usage;
  if (headroom > SOME_APP_DEFINED_VALUE){
    // Do memory-intensive operation.
  }
}
```

It helps to think of memory as just another device capability. That is, it's common for an app to test the availability of the device features it can use. Is there a compass on this device? Is there a forward-facing camera? Also, some features are available only in certain app states. For example, if a device has a microphone, it's almost always available to the app in the foreground, but typically not available to any background task. So it behooves the app to check availability at different times. In the same way, the app should be testing how much memory is available to it at any given time. The app can adapt to this by, for example, selecting different image resolutions, or different data transfer options, or even by completely enabling or disabling certain app features. Documentation for the MemoryManager API is at bit.ly/2bqepDL.

Memory Limits

What happens if an app hits its limit? Contrary to popular belief, in most cases, the Resource Manager doesn't terminate apps for out-of-memory conditions. Instead, if the app does something that would result in a memory allocation that would exceed its limit, the allocation fails. In some cases, the failure is surfaced to the app (as an OutOfMemoryException in a managed code app, or a null pointer in a native app). If this happens, the app can handle the failure. If not, the app will crash. Consider the following examples. DoSomething is allocating simple byte array memory in an infinite loop that will eventually result in an OutOfMemoryException, which the app can handle:

```
private void DoSomething(){
  List<byte[]> buffer = new List<byte[]>();
  try {
    while (true)
      buffer.Add(new byte[10 * 1024 * 1024]);
  }
  catch (OutOfMemoryException oomEx){
    // Handle the exception and recover.
  }
}
```

Conversely, DoAnother is using imaging APIs in an infinite loop that are internally allocating memory on the native heap for graphics data. This allocation is outside the app's direct control, and when it fails, it will almost certainly *not* propagate any exception that can be handled to the app and, therefore, the app will simply crash:

```
private void DoAnother(IRandomAccessStream stream){
  List<BitmapImage> bitmaps = new List<BitmapImage>();
  while (true){
    stream.Seek(0);
    BitmapImage bitmap = new BitmapImage();
    bitmap.SetSource(stream);
    bitmaps.Add(bitmap);
  }
}
```

The scenario is a little contrived, as no app would realistically expect to be able to create an infinite number of bitmaps, but the point is that some allocation failures are easily handled while others are not. You should handle OutOfMemoryExceptions when you can, and examine your app code for scenarios where memory is allocated outside your direct control; police these areas carefully to avoid failures. You're more likely to be successful handling exceptions for operations that allocate large amounts of memory—attempting to handle OutOfMemoryExceptions for small allocations is usually not worth the added complexity. It's also worth noting that an app can hit an OutOfMemoryException well below its limit if it's making very large allocations—and especially in managed code. This can arise as a result of address space fragmentation for your process. For example, the DoSomething method is allocating 10MB blocks, and it will hit OutOfMemoryException sooner than if it were allocating 1MB blocks. Finally, it must be said that the cases where your app can handle an OutOfMemoryException and continue in a meaningful way are rare; in practice, it's more often used as an opportunity to clean up, notify the user and then fail gracefully.

Figure 1 **The Two Broad Categories of App Task**

Category	Typical Examples	Description
Critical tasks	Foreground app activations and some important background tasks such as VoIP, background audio playback and any background task invoked directly by a foreground app.	These are effectively always guaranteed to run whenever requested (except in cases of extreme and unexpected system process activity).
Opportunistic tasks	Everything else.	These are only allowed to launch (or to continue to run) when there are sufficient available resources and there's no higher-priority task contending those resources. There are multiple finely grained priority levels within this category.

Using Task Priorities to Resolve Contention

The system arbitrates between competing task types by weighing the relative importance of each user scenario. For example, the system generally assigns a higher priority to the app with which the user is actively engaged, and a lower priority to background activity of which the user might even be completely unaware. Even among background tasks there are different priority levels. For example, VoIP and push notification tasks are typically higher priority than time-triggered tasks.

When the user launches an app, or when a trigger event tries to activate a background task, the Resource Manager checks to see if there are sufficient free resources for this request. If there are, the activation goes ahead. If not, it then examines all running tasks and

starts canceling (or in some cases rudely terminating) tasks from the lowest priority upward until it has freed enough resources to satisfy the incoming request.

Prioritization is finely nuanced, but everything falls into one of two broad priority categories, summarized in **Figure 1**.

Soft and Hard Memory Limits

Resource policy limits ensure that no one app can run away with all the memory on the device to the exclusion of other scenarios. However, one of the side effects is that a situation can arise where a task can hit its memory limit even though there might be free memory available in the system.

The Windows 10 Anniversary Update addresses this by relaxing the hard memory limits to *soft* limits. To best illustrate this, consider the case of extended execution scenarios. In previous releases, when an app is in the foreground it has, say, a 400MB limit (a fictitious value for illustration only), and when it transitions to the background for extended execution, policy considers it to be less important—plus it doesn't need memory for UI rendering—so its limit is reduced to perhaps 200MB. Resource policy does this to ensure that the user can successfully run another foreground app at the same time. However, in the case where the user *doesn't* run another foreground app (other than Start), or runs only a small foreground app, the extended execution app may well hit its memory limit and crash even though there's free memory available.

> So in the Windows 10 Anniversary Update, when the app transitions to extended execution in the background, even though its limit is reduced, *it's allowed to use more memory than its limit*.

So in Windows 10 Anniversary Update, when the app transitions to extended execution in the background, even though its limit is reduced, *it's allowed to use more memory than its limit*. In this way, if the system isn't under memory pressure, the extended execution app is allowed to continue, increasing the likelihood that it can complete its work. If the app does go over its limit, the MemoryManager API will report that its AppMemoryUsageLevel is OverLimit. It's important to consider that when an app is over-limit, it's at higher risk of getting terminated if the system comes under memory pressure. The exact behavior varies per platform: Specifically, on Xbox, an over-limit app has two seconds to get itself below its limit or it will be suspended. On all other platforms, the app can continue indefinitely unless and until there's resource pressure.

The net result of this change is that more tasks will be able to continue in the background more often than before. The only downside is that the model is slightly less predictable: Previously, a task that attempted to exceed its limit would always fail to allocate (and likely crash). Now, the allocation-failure-and-crash behavior doesn't *always* follow: The task will often be allowed to exceed its limit without crashing.

The Resource Manager raises the AppMemoryUsageIncreased event when an app's memory usage increases from any given level to a higher level, and conversely, the AppMemoryUsageDecreased event when it decreases a level. An app can respond to AppMemoryUsageIncreased by checking its level and taking appropriate action to reduce its usage:

```
private void OnUsageIncreased(object sender, object e){
  AppMemoryUsageLevel level = MemoryManager.AppMemoryUsageLevel;
  if (level == AppMemoryUsageLevel.OverLimit
    || level == AppMemoryUsageLevel.High){
    // Take action to reduce memory usage.
  }
  else if (level == AppMemoryUsageLevel.Medium){
    // Continue to monitor memory usage.
  }
}
```

Then, when it has successfully reduced its usage, it can expect to get a further notification that it has fallen to a safer level, via an AppMemoryUsageDecreased event:

```
private void OnUsageDecreased(object sender, object e){
  AppMemoryUsageLevel level = MemoryManager.AppMemoryUsageLevel;
  if (level == AppMemoryUsageLevel.Medium){
    // Back down at an acceptable level.
  }
  else if (level == AppMemoryUsageLevel.Low){
    // Can start memory-intensive operations.
  }
}
```

An app can also sign up for the AppMemoryUsageLimitChanging event, which the Resource Manager raises when it changes an app's limit. The OverLimit scenario deserves special handling, because of the associated change in priority. An app can listen to the notification event that's raised when the system changes its limit, so it can immediately take steps to reduce its memory consumption. For this scenario, you should use the old and new limit values passed in as payload of the event, rather than querying the AppMemoryUsageLevel directly:

```
private void OnMemoryLimitChanging(object sender,
  AppMemoryUsageLimitChangingEventArgs e){
  if (MemoryManager.AppMemoryUsage >= e.NewLimit){
    // Take urgent action to reduce usage.
  }
}
```

Extended execution is just one of the scenarios where the limit is changed. Another common scenario is where the app calls external app services—each of these will reduce the calling app's limit for the duration of the call. It's not always obvious when an app is calling an app service: For example, if the app uses a middleware library, this might implement some APIs as app services under the covers. Or, if the app calls into system apps, the same might happen; Cortana APIs are a case in point.

ProcessDiagnosticInfo API

Commit usage is the amount of virtual memory the app has used, including both physical memory and memory that has been paged out to the disk-backed pagefile. Working set is the set of memory pages in the app's virtual address space that's currently resident in physical memory. For a detailed breakdown of memory terminology, see bit.ly/2b5UwjL. The MemoryManager API exposes both a

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

NEW! 1&1 MANAGED CLOUD HOSTING

The best of both worlds!

Scalable on-demand and flexible server resources combined with a new high-performance hosting package: the **new 1&1 Managed Cloud Hosting is here!** Ideal for online projects which demand only the best availability, security and flexibility.

- ✓ **Dedicated resources**
- ✓ **20+ stack variations**
- ✓ **Managed by 1&1 experts**
- ✓ **Flexible scalability**
- ✓ **Deployed <1 minute**

starting at $9.99 per month*

Cloud Vendor Benchmark 2016
Rising Star Germany
experton GROUP an ISG business

intel XEON inside™

Trusted Performance.
Intel® Xeon® processors.

1 MONTH FLEXIBLE PAYMENT OPTIONS

1 CALL SPEAK WITH AN EXPERT 24/7

1 SECURE CHOICE FAIL-SAFE

1-844-296-2059

1&1®

1and1.com

*1&1 Managed Cloud Hosting starting at $9.99 per month. No minimum contract term. No setup fee.
1&1 Internet Inc., 701 Lee Road, Chesterbrook, PA 19087

GetAppMemoryReport and a GetProcessMemoryReport for commit metrics and working-set metrics, respectively. Don't be misled by the names of the properties—for example, in the AppMemoryReport class, the private commit used by the app is represented by PrivateCommitUsage (which seems obvious), whereas in the ProcessMemoryUsageReport class the same value is represented by PageFileSizeInBytes (which is a lot less obvious). Apps can also use a related API: Windows.System.Diagnostics.ProcessDiagnosticInfo. This provides low-level diagnostic information on a per-process basis, including memory diagnostics, CPU and disk-usage data. This is documented at bit.ly/2b1lokD. There's some overlap with the MemoryManager API, but there's additional information in ProcessDiagnosticInfo beyond what's available in MemoryManager. For example, consider an app that allocates memory, but doesn't immediately use it:

```
private List<byte[]> buffer = new List<byte[]>();

private void ConsumeMemory(){
  byte[] data = new byte[SOME_APP_DEFINED_VALUE];
  buffer.Add(data);
}
```

You could use the ProcessMemoryReport or ProcessMemoryUsageReport to get information about commit and working-set, including private (used only by this app), total (includes private plus shared working set), and peak (the maximum used during the current process's lifetime so far). For comparison, note that the memory usage reported by Task Manager is the app's private working-set:

```
private void GetDiagnostics(){
  Trace("commit={0:N}", MemoryManager.AppMemoryUsage);
  ProcessMemoryReport r1 = MemoryManager.GetProcessMemoryReport();
  Trace("private w/s={0:N}", r1.PrivateWorkingSetUsage);
  Trace("total w/s={0:N}", r1.TotalWorkingSetUsage);

  ProcessDiagnosticInfo info =
    ProcessDiagnosticInfo.GetForCurrentProcess();
  ProcessMemoryUsageReport r2 = info.MemoryUsage.GetReport();
  Trace("peak w/s={0:N}", r2.PeakWorkingSetSizeInBytes);
}
```

Each time the app calls its ConsumeMemory method, more commit is allocated, but unless the memory is used, it doesn't significantly increase the working set. It's only when the memory is used that the working set increases:

```
private void ConsumeMemory(){
  byte[] data = new byte[SOME_APP_DEFINED_VALUE];
  for (int i = 0; i < data.Length; i++)
    data[i] = 1;
  buffer.Add(data);
}
```

Most apps only need to focus on commit (which is what the Resource Manager bases its decisions on), but some more sophisticated apps might be interested in tracking working-set, also. Some apps, notably games and media-intensive apps, rapidly switch from one set of data to the next (think graphics buffers), and the more their data is in physical memory, the more they can avoid UI stuttering and tearing.

Also, you can think of memory as a closed ecosystem: It can be useful to track your working-set just to see how much pressure you're putting on the system as a whole. Certain system operations—such as creating processes and threads—require physical memory, and if your app's working-set usage is excessive this can degrade performance system-wide. This is particularly important on the desktop, where policy doesn't apply commit limits.

GlobalMemoryStatusEx API

From the Windows 10 Anniversary Update, apps also have available to them the Win32 GlobalMemoryStatusEx API. This provides some additional information beyond the Windows RT APIs, and while most apps will never need to use it, it has been provided for the benefit of UWP apps that are highly complex and have very finely tuned memory behaviors. To use this API you also need the MEMORYSTATUSEX struct, as shown in **Figure 2**.

Then, you can instantiate this struct and pass it to GlobalMemoryStatusEx, which will fill in the struct fields on return:

```
MEMORYSTATUSEX status = new MEMORYSTATUSEX();
if (GlobalMemoryStatusEx(status))
{
  Trace("TotalPhys={0:N}", status.ullTotalPhys);
  Trace("AvailPhys={0:N}", status.ullAvailPhys);
  Trace("TotalPageFile={0:N}", status.ullTotalPageFile);
  Trace("AvailPageFile={0:N}", status.ullAvailPageFile);
  // ... etc.
}
```

Again, don't be misled by the names of the fields. For example, if you're interested in the size of the pagefile, don't just look at ullTotalPageFile, because this actually represents the current maximum amount of commit, which includes both the pagefile and physical memory. So, what most folks understand as the pagefile size is computed by subtracting the ullTotalPhys value from the ullTotalPageFile value, like so:

```
ulong pageFile =
  status.ullTotalPageFile - status.ullTotalPhys;
```

Also note that ullTotalPhys is not the total amount of memory physically installed on the device. Rather, it's the amount of physical memory the OS has available to it at boot, which is always slightly less than the absolute total of physical memory.

Another interesting value returned is dwMemoryLoad, which represents the percentage of physical memory in use system-wide. In some environments it's important for an app's memory usage to be mostly in physical memory, to avoid the disk I/O overhead of using the pagefile. This is especially true for games and media apps—and critically important for Xbox and HoloLens apps.

Remember this is a Win32 API so it will return information that doesn't account for the UWP sandbox and, in particular, it has no knowledge of resource policy. So, for example, the value returned in

Figure 2 **Importing the GlobalMemoryStatusEx Win32 API**

```
[StructLayout(LayoutKind.Sequential)]
private class MEMORYSTATUSEX
{
  public uint dwLength;
  public uint dwMemoryLoad;
  public ulong ullTotalPhys;
  public ulong ullAvailPhys;
  public ulong ullTotalPageFile;
  public ulong ullAvailPageFile;
  public ulong ullTotalVirtual;
  public ulong ullAvailVirtual;
  public ulong ullAvailExtendedVirtual;
  public MEMORYSTATUSEX()
  {
    dwLength = (uint)Marshal.SizeOf<MEMORYSTATUSEX>();
  }
}

[DllImport("kernelbase.dll", SetLastError = true)]
[return: MarshalAs(UnmanagedType.Bool)]
private static extern bool GlobalMemoryStatusEx(
  [In, Out] MEMORYSTATUSEX lpBuffer);
```

LIVE! 360
TECH EVENTS WITH PERSPECTIVE

SPECIAL PULL-OUT SECTION

Orlando 2016

ROYAL PACIFIC RESORT AT UNIVERSAL
DECEMBER 5-9

The Ultimate Education Destination

Live! 360℠ is a unique conference where the IT and Developer community converge to debate leading edge technologies and educate themselves on current ones. These six co-located events incorporate knowledge transfer and networking, along with finely tuned education and training, as you create your own custom conference, mixing and matching sessions and workshops to best suit your needs. **Choose the ultimate education destination: Live! 360.**

LIVE360EVENTS.COM

EVENT PARTNERS
Microsoft
Magenic

PLATINUM SPONSORS
HYPERGRID
SMARTBEAR

GOLD SPONSORS
GOVERLAN
IDERA

SILVER SPONSOR
DEVIQ

SUPPORTED BY
ADTMAG
msdn magazine
Visual Studio MAGAZINE
Redmond Channel Partner
Redmond MAGAZINE
VIRTUALIZATION REVIEW

PRODUCED BY
1105 MEDIA
YOUR GROWTH, OUR BUSINESS

LIVE! 360
TECH EVENTS WITH PERSPECTIVE

Live! 360 Agenda-at-a-Glance

Visual Studio LIVE!

VISUAL STUDIO LIVE! TRACKS

ALM / DevOps | Cloud Computing | Mobile Client | Software Practices | Visual Studio / .NET Framework | Web Client | Web Server | Windows Client

SQL Server LIVE!

SQL SERVER LIVE! TRACKS

BI, Big Data, Data Analytics, and Data Visualization | SQL Server Administration & Maintenance | SQL Server in the Cloud | SQL Server for Developers | SQL Server Performance Tuning and Optimization

TECHMENTOR

TECHMENTOR TRACKS

Client | DevOps | IT Soft Skills | The Real Cloud | Security | Server / Datacenter

START TIME	END TIME	Visual Studio Live! Pre-Conference: Sunday, December 4, 2016	SQL Server Live! Pre-Conference: Sunday, December 4, 2016	TechMentor Pre-Conference: Sunday, December 4, 2016
5:00 PM	8:00 PM	Pre-Conference Registration • Royal Pacific Resort Conference Center	Pre-Conference Registration • Royal Pacific Resort Conference Center	
6:00 PM	9:00 PM	Dine-A-Round Dinner @ Universal CityWalk *(6:00pm—Meet at Conference Registration Desk to walk over with the group)*	Dine-A-Round Dinner @ Universal CityWalk *(6:00pm—Meet at Conference Registration Desk to walk over with the group)*	

START TIME	END TIME	Visual Studio Live! Pre-Conference Workshops: Monday, December 5, 2016	SQL Server Live! Pre-Conference Workshops: Monday, December 5, 2016	TechMentor Pre-Conference Workshops: Monday, December 5, 2016
8:00 AM	5:00 PM	**VSM01** - Workshop: Distributed Cross-Platform Application Architecture - *Rockford Lhotka & Jason Bock* **VSM02** - Workshop: Service Oriented Technologies—Designing, Developing, & Implementing WCF and the Web API - *Miguel Castro* **VSM03** - Workshop: DevOps in a Day - *Brian Randell*	**SQM01** - Workshop: Performance Tune SQL Server: Query Optimizer, Indexes, the Plan Cache, and Execution Plans - *Bradley Ball* **SQM02** - Workshop: SQL Server 2016 for Developers - *Leonard Lobel*	**TMM01** - Workshop: Sixty-seven VMware vSphere Tricks That'll Pay for This Conference! - *Greg Shields* **TMM02** - Workshop: Demystifying the Blue Screen of Death - Bring Your Own Laptop Hands-On Lab (BYOL - HOL) - *Bruce Mackenzie-Low*
5:00 PM	6:00 PM	EXPO Preview		
6:00 PM	7:00 PM	**LIVE! 360 KEYNOTE:** To Be Announced		

START TIME	END TIME	Visual Studio Live! Day 1: Tuesday, December 6, 2016	SQL Server Live! Day 1: Tuesday, December 6, 2016	TechMentor Day 1: Tuesday, December 6, 2016
8:00 AM	9:00 AM	**Visual Studio Live! Keynote: Topic to be announced** – *Tim Sneath, Principal Lead Program Manager, Visual Studio Platform, Microsoft*	**SQL SERVER LIVE! KEYNOTE: The RDBMS is Dead. Long live the RDBMS!** *– Buck Woody, Senior Technical Specialist, Machine Learning and Data Science Team, Microsoft*	**TECHMENTOR KEYNOTE: Sweet Sixteen, or Just Server 2012R3? A Glance at the Awesome, the Irritating, the Improved and the Expensive in Server 2016** - *Mark Minasi, IT Consultant, Author, Speaker, MR&D*
9:00 AM	9:30 AM			Networking Break • Visit the EXPO
9:30 AM	10:45 AM	**VST01** - Building Applications with ASP.NET Core - *Scott Allen* **VST02** - Busy .NET Developer's Guide to Swift - *Ted Neward* **VST03** - What's New in Azure v2 - *Eric D. Boyd* **VST04** - Real World Scrum with Team Foundation Server 2015 & Visual Studio Team Services - *Benjamin Day*	**SQT01** - Performance Tuning and Monitoring for Virtualized Database Servers - *Thomas LaRock* **SQT02** - What's New in SQL Server 2016 - *Leonard Lobel* **SQT03** - Power BI: Analytics for Desktop, Mobile and Cloud - *Andrew Brust*	**TMT01** - Secure Access Everywhere! Implementing DirectAccess in Windows Server 2016 - *Richard Hicks* **TMT02** - Linux on Azure for the Microsoft Specialist - *Timothy Warner* **TMT03** - The Absolute Beginner's Guide to Advanced Certificate Services - *Greg Shields*
11:00 AM	12:15 PM	**VST05** - Richer MVC Sites with Knockout JS - *Miguel Castro* **VST06** - Busy .NET Developer's Guide to Native iOS - *Ted Neward* **VST07** - Overview of Power Apps - *Nick Pinheiro* **VST08** - Get Good at DevOps: Feature Flag Deployments with ASP.NET, WebAPI, & JavaScript - *Benjamin Day*	**SQT04** - U-SQL Query Execution and Performance Tuning - *Michael Rys* **SQT05** - Implementing Data Protection and Security in SQL Server 2016 - *Steve Jones* **SQT06** - Seeking the Perfect Hybrid: On-Prem Data with Reports and Dashboards in Power BI - *William E. Pearson III*	**TMT04** - DirectAccess Troubleshooting Deep Dive - *Richard Hicks* **TMT05** - Container Technology and its Impact on Datacenter and Cloud Management - Neil Peterson **TMT06** - Master Camtasia and Build Your Own Training in 75 Minutes or Less - *Greg Shields*
12:15 PM	2:00 PM			Lunch • Visit the EXPO
2:00 PM	3:15 PM	**VST09** - WCF & Web API: Can We All Just Get Along?!? - *Miguel Castro* **VST10** - Creating Great Looking Android Applications Using Material Design - *Kevin Ford* **VST11** - Introduction to Next Generation of Azure PaaS—Service Fabric and Containers - *Vishwas Lele* **VST12** - To Be Announced	**SQT07** - New Performance Tuning and Security Features in SQL Server 2016 - *Thomas LeBlanc* **SQT08** - Introduction to Elastic Azure SQL Database - *Bradley Ball* **SQT09** - To Be Announced	**TMT07** - Windows as a Service Explained: Really, I've Got to Upgrade Every Year? - *Mark Minasi* **TMT08** - To Be Announced **TMT09** - Getting Started with Nano Server - *Jeffery Hicks*
3:15 PM	4:15 PM			Networking Break • Visit the EXPO
4:15 PM	5:30 PM	**VST13** - Busy Developer's Guide to Chrome Development - *Ted Neward* **VST14** - Using Visual Studio Tools for Apache Cordova to Create MultiPlatform Applications - *Kevin Ford* **VST15** - Cloud Oriented Programming - *Vishwas Lele* **VST16** - Bringing DevOps to the Database - *Steve Jones*	**SQT10** - Performance Tuning and Troubleshooting Azure SQL Database - *Bradley Ball* **SQT11** - No Schema, No Problem!: Introduction to Azure DocumentDB - *Leonard Lobel* **SQT12** - Getting Started with Data Analysis eXpressions (DAX) in Analysis Services Tabular 2016 - *William E. Pearson III*	**TMT10** - Troubleshooting Client Communications with Wireshark - *Timothy Warner* **TMT11** - Azure Point-to-Site VPN Suck! Fix It with Win2012R2 VPN in the Cloud - *Richard Hicks* **TMT12** - Implementing Hyper-V Failover Clusters in Windows Server 2016 - *Bruce Mackenzie-Low*
5:30 PM	7:30 PM			Exhibitor Reception

START TIME	END TIME	Visual Studio Live! Day 2: Wednesday, December 7, 2016	SQL Server Live! Day 2: Wednesday, December 7, 2016	TechMentor Day 2: Wednesday, December 7, 2016
8:00 AM	9:15 AM	**VSW01** - Moving from Angular 1 to Angular 2 - *Ben Dewey* **VSW02** - The Future of Mobile Application Search - *James Montemagno* **VSW03** - Managing Enterprise and Consumer Identity with Azure Active Directory - *Nick Pinheiro* **VSW04** - Improving Performance in .NET Applications - *Jason Bock*	**SQW01** - Inside the SQL Server Query Optimizer: SQL 2014 & 2016 - *Bradley Ball* **SQW02** - SQL Server 2016 Encryption - *Thomas LaRock* **SQW03** - Big Data with Hadoop, Spark and Azure HDInsight - *Andrew Brust*	**TMW01** - Creating Advanced Functions in PowerShell - *Michael Wiley* **TMW02** - Implementing Azure AD for Hybrid Identity - *Timothy Warner* **TMW03** - Managing Windows 10 Using the MDM Protocol and ConfigMgr - *Steven Rachui*
9:30 AM	10:45 AM	**VSW05** - Getting Started with Aurelia - *Brian Noyes* **VSW06** - Building Connected and Disconnected Mobile Applications - *James Montemagno* **VSW07** - Practical Internet of Things for the Microsoft Developer - *Eric D. Boyd* **VSW08** - I'll Get Back to You: Understanding Task, Await, and Asynchronous Methods - *Jeremy Clark*	**SQW04** - Does Your Performance Tuning Need a 12-Step program? - *Janis Griffin* **SQW05** - Master Data Management with Data Quality (DQS) and Master Data Services (MDS) in SQL Server 2016 - *Thomas LeBlanc* **SQW06** - Getting Started with PolyBase in SQL Server 2016 - *Edwin Sarmiento*	**TMW04** - Creating Class-Based PowerShell Tools - *Jeffery Hicks* **TMW05** - Fully Integrated Azure Resource Manager Deployments - *Neil Peterson* **TMW06** - Building Applications in ConfigMgr—Tips and Tricks - *Steven Rachui*
10:45 AM	11:15 AM	Networking Break • Visit the EXPO		
11:15 AM	12:15 PM	**LIVE! 360 KEYNOTE: To Be Announced**		
12:15 PM	1:45 PM	Birds-of-a-Feather Lunch • Visit the EXPO		
1:45 PM	3:00 PM	**VSW09** - Living in a Command Line Web Development World (NPM, Bower, Gulp, and More) - *Ben Dewey* **VSW10** - Understanding the Windows Desktop App Development Landscape - *Brian Noyes* **VSW11** - To Be Announced **VSW12** - Learn to Love Lambdas (and LINQ, Too) - *Jeremy Clark*	**SQW07** - Configuring SQL Server for Performance—Like a Microsoft Certified Master - *Thomas LaRock* **SQW08** - To Be Announced **SQW09** - Big Data's Missing V: Visualization. How Do You BigViz Your Big Data? - *Jennifer Stirrup*	**TMW07** - Creating WPF-Based Graphical PowerShell Tools - *Jeffery Hicks* **TMW08** - To Be Announced **TMW09** - SysInternals Tools: Process Explorer and Process Monitor - *Sami Laiho*
3:00 PM	4:00 PM	Networking Break • Visit the EXPO • Expo Raffle @ 3:30 p.m.		
4:00 PM	5:15 PM	**VSW13** - Securing Client JavaScript Apps - *Brian Noyes* **VSW14** - Let's Write a Windows 10 App: A Basic Introduction to Universal Apps - *Billy Hollis* **VSW15** - ARM Yourself for Azure Success - *Esteban Garcia* **VSW16** - Continuous Delivery on Azure: A/B Testing, Canary Releases, and Dark Launching - *Marcel de Vries*	**SQW10** - Performance in 60 Seconds—SQL Tricks Everybody MUST Know - *Pinal Dave* **SQW11** - Welcome to the 2016 Query Store! - *Janis Griffin* **SQW12** - Improve Enterprise Reporting with SQL Server Analysis Services - *Thomas LeBlanc*	**TMW10** - Harvesting the Web: Using PowerShell to Scrape Screens, Exploit Web Services, and Save Time - *Mark Minasi* **TMW11** - In-Depth Introduction to Docker - *Neil Peterson* **TMW12** - War Driving: How it Happens, How to Protect Yourself - *Dale Meredith*
8:00 PM	10:00 PM	Live! 360 Dessert Luau		

START TIME	END TIME	Visual Studio Live! Day 3: Thursday, December 8, 2016	SQL Server Live! Day 3: Thursday, December 8, 2016	TechMentor Day 3: Thursday, December 8, 2016
8:00 AM	9:15 AM	**VSH01** - Build Real-Time Websites and Apps with SignalR - *Rachel Appel* **VSH02** - Cognitive Services: Building Smart Applications with Computer Vision - *Nick Landry* **VSH03** - C# Best Practices - *Scott Allen* **VSH04** -Application Insights: Measure Your Way to Success - *Esteban Garcia*	**SQH01** - Stretching SQL Server Failover Clustered Instances & Availability Groups to a Disaster Recovery Site - *Edwin Sarmiento* **SQH02** - Production SQL Server 2016—Lessons from the Field - *Joseph D'Antoni* **SQH03** - Agile Analytics with AzureML and R - *Jennifer Stirrup*	**TMH01** - PowerShell and Workflow—Magic Together! - *Michael Wiley* **TMH02** - Pen-Testing Like an IT Superhero - *Dale Meredith* **TMH03** - Windows Clusters for Beginners: From Highly Fearful to Highly Reliable in 75 Minutes! - *Mark Minasi*
9:30 AM	10:45 AM	**VSH05** - HTTP/2: What You Need to Know - *Robert Boedigheimer* **VSH06** - Building Business Apps on the Universal Windows Platform - *Billy Hollis* **VSH07** - Debugging Your Way Through .NET with Visual Studio 2015 - *Ido Flatow* **VSH08** - The Ultimate Intro to Docker for Developers - *Adam Tuliper*	**SQH04** - Indexes—The Good, Bad and Ugly - *Pinal Dave* **SQH05** - Would You Just Load Already?! Maximizing Your SSIS Data Load - *Chris Bell* **SQH06** - A Blueprint for Business Intelligence with SQL Server 2016 - *Jennifer Stirrup*	**TMH04** - You're Writing Your PowerShell Functions Wrong. Stop It. - *Don Jones* **TMH05** - Mobile Devices and Security: The Bane of the IT Superhero - *Dale Meredith* **TMH06** - Office 365 Risk Mitigation - *J. Peter Bruzzese*
11:00 AM	12:15 PM	**VSH09** - TypeScript and ES2015 JumpStart - *John Papa* **VSH10** - HoloLens - *Billy Hollis & Brian Randell* **VSH11** - Exploring Microservices in a Microsoft Landscape - *Marcel de Vries* **VSH12** - Automated UI Testing for iOS and Android Mobile Apps - *James Montemagno*	**SQH07** - Securing Your Database with SQL Server 2016 Features - *Edwin Sarmiento* **SQH08** - Statistics and Cardinality—How They Work Together to Find Data Efficiently - *Chris Bell* **SQH09** - New and Shiny: The All-New SQL Server Reporting Services 2016 - *Tim Mitchell*	**TMH07** - PowerShell Desired State Configuration (DSC) for the IT Ops Guy - *Jason Helmick* **TMH08** - Understanding Windows 10/2016's Super Security: VSM, Credential Guard, Trustlets and More - *Mark Minasi* **TMH09** - Evolving as an IT Pro - *J. Peter Bruzzese*
12:15 PM	1:30 PM			Lunch on the Lanai
1:30 PM	2:45 PM	**VSH13** - All Your Tests Are Belong To Us - *Rachel Appel* **VSH14** - Developing Awesome 3D Apps with Unity and C# - *Adam Tuliper* **VSH15** - Unit Testing Makes Me Faster: Convincing Your Boss, Your Co-Workers, and Yourself - *Jeremy Clark* **VSH16** -Writing Maintainable, X-Browser Automated Tests - *Marcel de Vries*	**SQH10** - Secrets of SQL Server—Database Worst Practices - *Pinal Dave* **SQH11** - Powerful T-SQL Improvements that Reduce Query Complexity - *Hugo Kornelis* **SQH12** - Testing SSIS Packages - *Tim Mitchell*	**TMH10** - PowerShell Unplugged: Stump Don - *Don Jones* **TMH11** - Facing Increasing Malware Threats and a Growing Trend of BYOD with a New Approach of PC Security - *Yung Chou* **TMH12** - The Labyrinth of Exchange Migration Options - *J. Peter Bruzzese*
3:00 PM	4:15 PM	**VSH17** - SASS and CSS for Developers - *Robert Boedigheimer* **VSH18** - From Oculus to HoloLens: Building Virtual & Mixed Reality Apps & Games - *Nick Landry* **VSH19** - User Experience Case Studies—Good and Bad - *Billy Hollis* **VSH20** - Debugging the Web with Fiddler - *Ido Flatow*	**SQH13** - Hacking Expose—Using SSL to Secure SQL Server Connections - *Chris Bell* **SQH14** - T-SQL User-Defined Functions, or: Bad Performance Made Easy - *Hugo Kornelis* **SQH15** - Essential Architecture for BI in a Virtual Environment - *Joseph D'Antoni*	**TMH13** - Building Highly Available and Secure DSC Pull Servers - *Jason Helmick* **TMH14** - Penetration Tests in Real Life - *Sami Laiho* **TMH15** - What's IoT Got to Do with IT Pros? A Lot! - *Yung Chou*
4:30 PM	5:30 PM			**Live! 360 Conference Wrap-Up** - *Andrew Brust (Moderator), Andrew Connell, Don Jones, Rockford Lhotka, Matthew McDermott, Brian Randell, & John K. Waters*

START TIME	END TIME	Visual Studio Live! Post-Conference Workshops: Friday, December 9, 2016	SQL Server Live! Post-Conference Workshops: Friday, December 9, 2016	TechMentor Post-Conference Workshops: Friday, December 9, 2016
8:00 AM	5:00 PM	**VSF01** - Workshop: Angular 2 Bootcamp - *John Papa* **VSF02** - Workshop: Building Modern Web Apps with Azure - *Eric D. Boyd & Brian Randell*	**SQF01** - Workshop: Design and Implement SQL Server HA/DR Hybrid Solutions with Microsoft Azure - *Edwin Sarmiento* **SQF02** - Workshop: Big Data, Analytics and NoSQL: Everything You Wanted to Learn But Were Afraid to Ask - *Andrew Brust*	**TMF01** - Workshop: Boost Your IT Career, 2017 Edition: The Don and Jason Show - *Don Jones and Jason Helmick* **TMF02** - Workshop: BlackBelt—Windows Security Internals - *Sami Laiho*

Speakers and sessions subject to change

ROYAL PACIFIC RESORT AT UNIVERSAL ORLANDO

DEC 5-9

Orlando 2016

Office & SharePoint LIVE!

OFFICE & SHAREPOINT LIVE! TRACKS

SharePoint On-Premises Infrastructure Management and Administration	SharePoint, Office 365 and the Cloud	High-Value SharePoint Workloads: Social, Search, BI, Workflow, and Business Process Automation	Developing for Office, Office 365 and SharePoint

Office & SharePoint Live! Pre-Conf. Sun., Dec. 4, 2016

Modern Apps LIVE!

MODERN APPS LIVE! TRACK

Presented in Partnership with: Magenic

MAL! Pre-Conf.: Sun., Dec. 4

APPDEV TRENDS NEW!

APP DEV TRENDS TRACKS

Agile	Container-ization	Continuous Integration	Java	Mobile	Cloud

ADT Pre-Conf.: Sun., Dec. 4

Pre-Conference Registration • Royal Pacific Resort Conference Center

Dine-A-Round Dinner @ Universal CityWalk *(6:00pm—Meet at Conference Registration Desk to walk over with the group)*

Office & SharePoint Live!: Mon., Dec. 5, 2016

8:00 AM – 12:00 PM: OSM01 - Workshop: Installing and Configuring SharePoint Server 2016 - *Vlad Catrinescu*

8:00 AM – 12:00 PM: OSM02 - Workshop: A Beginner's Guide to Client Side Development in SharePoint - *Mark Rackley*

1:00 PM – 5:00 PM: OSM03 - Workshop: Upgrade to SharePoint 2016 - *Matthew McDermott*

1:00 PM – 5:00 PM: OSM04 - Workshop: Building Clients Side Applications in Office 365 with the New SharePoint Framework - *Andrew Connell*

MAL!: Monday, Dec. 5

MAM01 - Workshop: Building Modern Mobile Apps - *Brent Edwards & Kevin Ford*

ADT: Pre-Conference Workshops: Monday, Dec. 5

ADM01 - Workshop: Building Teams - *Steve Green*

ADM02 - Workshop: One Codebase to Rule Them All: Xamarin - *Fabian Williams*

Office & SharePoint Live! Day 1: Tues., Dec. 6

OFFICE & SHAREPOINT LIVE! KEYNOTE: The Modern Workplace and the SharePoint Revolution - *Chris Bortlik, Collaboration Solution Architect, Microsoft*

OST01 - What's New in SharePoint 2016 for IT Pros - *Vlad Catrinescu*

OST02 - To Be Announced

OST03 - Optimizing SQL Server for SharePoint - *Brian Alderman*

OST04 - How It Works: Office 365 and the Microsoft Graph - *Bill Ayers*

OST05 - IT Pros Guide to Managing SharePoint Search - *Matthew McDermott*

OST06 - To Be Announced

OST07 - Integrating Office Online Server with SharePoint - *Brian Alderman*

OST08 - Introduction to the Office Dev PnP Core Library - *Rob Windsor*

MAL! Day 1: Tues., Dec. 6

MAL! KEYNOTE: Topic TBA – *Tim Sneath, Principal Lead Program Manager, Microsoft*

MAT01 - Modern App Development: Transform How You Build Web and Mobile Software - *Rockford Lhotka*

MAT02 - Architecture: The Key to Modern App Success - *Brent Edwards*

MAT03 - Manage Distributed Teams with Visual Studio Team Services and Git - *Brian Randell*

MAT04 - Focus on the User Experience #FTW - *Anthony Handley*

ADT Day 1: Tues., Dec. 6

APP DEV TRENDS KEYNOTE: You Are the Future of Enterprise Java! – *Reza Rahman, Speaker, Author, Consultant*

ADM01 - Hacking Technical Debt - *Steve Green*

ADT02 - Java 8 Lambdas and the Streaming API - *Michael Remijan*

ADT03 - Are You A SOLID Coder? - *Steve Green*

ADT04 - PrimeFaces 5: Modern UI Widgets for Java EE - *Kito Mann*

ADT05 - Agile Architecture - *Steve Green*

ADT06 - Full Stack Java with JSweet, Angular 2, PrimeNG, and JAX-RS - *Kito Mann*

ADT07 - Crafting Innovation - *Steve Green*

ADT08 - Who's Taking Out the Garbage? How Garbage Collection Works in the VM - *Kito Mann*

Office & SharePoint Live! Day 2: Wed., Dec. 7

OSW01 - PowerShell for Office 365 - *Vlad Catrinescu*

OSW02 - Become a Developer Hero by Building Office Add-ins - *Bill Ayers*

OSW03 - Managing Data Recovery in SharePoint - *Brian Alderman*

OSW04 - Utilizing jQuery in SharePoint—Get More Done Faster - *Mark Rackley*

OSW05 - Scripting SharePoint 2016 Tasks with PowerShell - *Ben Stegink*

OSW06 - Customizing Your SharePoint Forms Without Third Party Applications - *Mark Rackley*

OSW07 - Learn Best Practices for Managing and Administering SharePoint Online and OneDrive for Business - *Chris Bortlik*

OSW08 - Using the Office UI Fabric - *Paul Schaeflein*

MAL! Day 2: Wed., Dec. 7

MAW01 - DevOps, Continuous Integration, the Cloud, and Docker - *Dan Nordquist*

MAW02 - Mobile Panel - *Kevin Ford, Rockford Lhotka, James Montemagno, & Ryan J. Salva*

MAW03 - C# Everywhere: How CSLA .NET Enables Amazing Cross-Platform Code Reuse - *Rockford Lhotka*

MAW04 - Coding for Quality and Maintainability - *Jason Bock*

ADT Day 2: Wed., Dec. 7

ADW01 - Stop Killing Requirements! - *Melissa Green*

ADW02 - Migrating Customers to Microsoft Azure: Lessons Learned From the Field - *Ido Flatow*

ADW03 - Meeting-Free Software Development in Distributed Teams - *Yegor Bugayenko*

ADW04 - The Essentials of Building Cloud-Based Web Apps with Azure - *Ido Flatow*

ADW05 - Introduction to Microsoft Office Graph - *Fabian Williams*

ADW06 - Building IoT and Big Data Solutions on Azure - *Ido Flatow*

ADW07 - As You Think About Azure Databases, Think About DocumentDb - *Fabian Williams*

ADW08 - Where Does JavaScript Belong in the App Store? - *Jordan Matthiesen*

Office & SharePoint Live! Day 3: Thurs., Dec. 8

OSH01 - Implementing and Managing Office 365 - *Ben Stegink*

OSH02 - Enterprise JavaScript Development Patterns - *Rob Windsor*

OSH03 - Five Business Challenges of Hybrid Search in SharePoint 2016 and Office 365 - *Agnes Molnar*

OSH04 - Made for Mobile—Let the Microsoft Graph Power Your Mobile Apps! - *Bill Ayers*

OSH05 - To the Cloud! Using IaaS as a Hosting Provider for SharePoint - *Scott Hoag & Dan Usher*

OSH06 - Introduction to the SharePoint Client Object Model and REST API - *Rob Windsor*

OSH07 - Findability in YOUR Organization - *Agnes Molnar*

OSH08 - Leveraging Angular2 to Build Office Add-ins - *Andrew Connell*

OSH09 - Setting Up Directory Synchronization for Office 365 - *Scott Hoag & Dan Usher*

OSH10 - Webhooks in Office 365 - *Paul Schaeflein*

MAL! Day 3: Thurs., Dec. 8

MAH01 - Modern Mobile Development: Build a Single App For iOS & Android with Xamarin Forms - *Kevin Ford*

MAH02 - Universal Windows Development: UWP for PC, Tablet & Phone - *Brent Edwards*

MAH03 - Modern Web Development: ASP.NET MVC and Web API - *Allen Conway*

MAH04 - Modern Web Development: Building a Smart Web Client with TypeScript and Angular2 - *Allen Conway*

MAH05 - Using All That Data: Power BI to the Rescue - *Scott Diehl*

ADT Day 3: Thurs., Dec. 8

ADH01 - From VMs to Containers: Introducing Docker Containers for Linux and Windows Server - *Ido Flatow*

ADH02 - Continuous Testing in a DevOps World - *Wayne Ariola*

ADH03 - CQRS 2.0 - Commands, Actors, and Events...Oh My! - *David Hoerster*

ADH04 - Microservices as Chat Bots are the Future - *Yegor Bugayenko*

ADH05 - The Curious Case for the Immutable Object - *David Hoerster*

ADH06 - Continuous Integration May Have Negative Effects - *Yegor Bugayenko*

ADH07 - To Be Announced

ADH08 - Mobile DevOps Demystified with Xamarin, VSTS and HockeyApp - *Roy Cornelissen*

ADH09 - Get Started with Microsoft PowerApps - *Fabian Williams*

ADH10 - Overcoming the Challenges of Mobile Development in the Enterprise - *Roy Cornelissen*

O&SPL! Post-Conf. Workshop: Fri., Dec. 9

OSF01 - Workshop: 10 Steps to be Successful with Enterprise Search - *Agnes Molnar*

MAL!: Fri., Dec. 9

MAF01 - Workshop: Modern App Deep Dive: Xamarin, Responsive Web, UWP, CSLA .NET - *Jason Bock, Allen Conway, Brent Edwards & Kevin Ford*

ADT Post-Conf. Workshop: Fri., Dec. 9

ADF01 - Workshop: To Be Announced

REGISTER NOW AT LIVE360EVENTS.COM

Connect with Live! 360

twitter.com/live360
@live360

facebook.com
Search "Live 360"

linkedin.com
Join the "Live! 360" group!

REGISTER BY OCTOBER 5 AND SAVE $400!

Use promo code L360TipIn

Scan the QR code to register or for more event details.

Take the Tour

Six events, 25+ tracks, and literally hundreds of sessions to choose from – mix and match sessions to create your own, custom event line-up – it's like no other conference available today.

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

Visual Studio Live!: VSLive!™ is a victory for code, featuring unbiased and practical development training on the Microsoft Platform.

SQL Server LIVE!
TRAINING FOR DBAs AND IT PROS

SQL Server Live!: This conference will leave you with the skills needed to Lead the Data Race, whether you are a DBA, developer, IT Pro, or Analyst.

TECHMENTOR
IN-DEPTH TRAINING FOR IT PROS

TechMentor℠: This is IT training that finishes first, with zero marketing speak on topics you need training on now, and solid coverage on what's around the corner.

Office & SharePoint LIVE!
ON-PREMISE, CLOUD & CROSS-PLATFORM TRAINING

Office & SharePoint Live!: Provides leading-edge knowledge and training for SharePoint both on-premises and in Office 365 to maximizethe business value.

Modern Apps LIVE!
MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT

Modern Apps Live!: Presented in partnership with Magenic, this unique conference leads the way to learning how to architect, design and build a complete Modern App.

NEW! APPDEV TRENDS
ENTERPRISE FOCUSED. CODE DRIVEN.

App Dev Trends: This new technology conference focuses on the makers & maintainers of the software that power organizations in nearly every industry in the world– in other words, enterprise software professionals!

LIVE! 360
TECH EVENTS WITH PERSPECTIVE

Featured Speakers

Live! 360 brings you over 90 speakers; some of the best and brightest experts in the industry.

VIEW THE FULL LIST OF SPEAKERS AT LIVE360EVENTS.COM

Visual Studio LIVE!


Rachel Appel


Jason Bock


Billy Hollis


Miguel Castro


Marcel de Vries


Adam Tuliper


Nick Landry


Brian Noyes


John Papa


Brian Randell

SQL Server LIVE!


Andrew Brust


Pinal Dave


Edwin Sarmiento


Leonard Lobel


Janis Griffin

TECHMENTOR


Jeffery Hicks


Don Jones


Sami Laiho


Mark Minasi


Greg Shields

Office & SharePoint LIVE!


Bill Ayers


Mark Rackley


Andrew Connell


Matthew McDermott


Agnes Molnar

Modern Apps LIVE!


Allen Conway


Brent Edwards


Kevin Ford


Scott Diehl


Rockford Lhotka

APPDEV TRENDS NEW!


Steven Green


Melissa Green


Kito D Mann


Fabian Williams


Yegor Bugayenko

LIVE360EVENTS.COM

REGISTER BY OCTOBER 5 AND **SAVE $400!**
Use promo code L360TipIn

CONNECT WITH LIVE! 360
twitter.com/@live360events
facebook.com – Search "Live 360"
linkedin.com – Join the "Live 360" group!

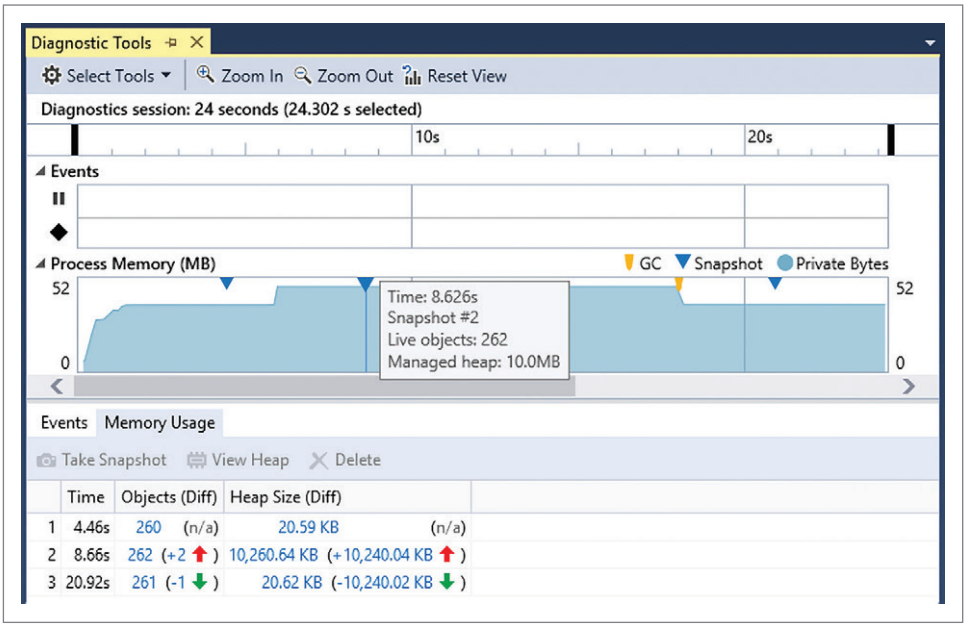


Figure 3 **Analyzing Process Memory in the Visual Studio Diagnostic Tools**

ullAvailPhys is the amount of physical memory currently available on the system, but this doesn't mean that this memory is actually available to the app. On all platforms apart from the desktop, it's likely to be significantly more than the amount of memory the current UWP app will actually be allowed to use, because its commit usage is constrained by policy, regardless of available physical memory.

For most apps, the MemoryManager API gives you all you need, and all the metrics that you can directly and easily influence in your app. ProcessDiagnosticInfo and GlobalMemoryStatusEx include some additional information that you can't directly influence, but which a more sophisticated app might want to pivot off for logic decisions, for profiling during development, and for telemetry purposes.

Visual Studio Diagnostics Tools

The memory diagnostic tool in Visual Studio 2015 updates its report in real time during debugging. You can turn this on while in a debug session by selecting the Debug menu, and then Show Diagnostic Tools, as shown in **Figure 3**.

The live graph in the Process Memory window tracks private commit, which corresponds to the AppMemoryReport.PrivateCommitUsage and the ProcessMemoryUsageReport.PageFileSizeInBytes. It doesn't include shared memory, so it represents only part of the metric reported in MemoryManager.AppMemoryUsage, for example. Note that if you hover over any point in the graph, you'll get a tooltip with usage data for that point in time.

You can also use the tool to take snapshots for more detailed comparisons. This is especially useful if you're trying to track down a suspected memory leak. In the following example, the app has two methods, one that allocates memory (simulating a leak) and the other that's naively attempting to release that memory:

```
private List<byte[]> buffer = new List<byte[]>();

private void SimulateLeak(){
  buffer.Add(new byte[10 * 1024 * 1024]);
}

private void CleanUp(){
  buffer.Clear();
}
```

A glance at the memory graph will show that the memory isn't actually getting released at all. In this example, the simplest fix is to force a garbage collection. Because collection is generational, in scenarios where you have complex object trees (which this example doesn't), you might need to make two collection passes, and also wait for the collected objects' finalizers to run to completion. If your app is allocating large objects, you can also set GCLargeObjectHeapCompactionMode to compact the Large Object Heap when a collection is made. The Large Object Heap is used for objects greater than 80KB; it's rarely collected unless forced; and even when collected, it can leave heap fragmentation. Forcing it to be compacted will increase your app's chances of

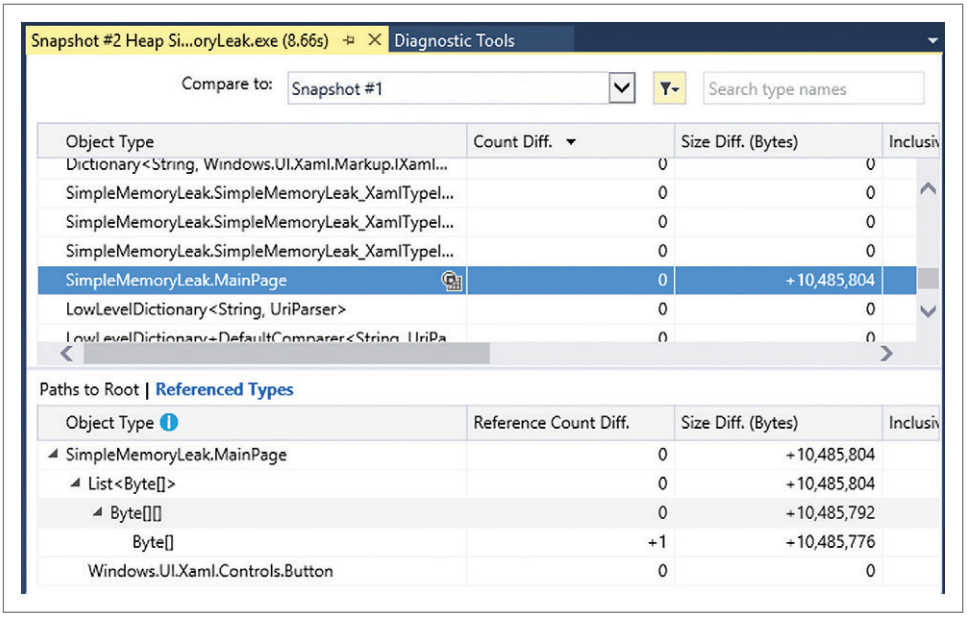


Figure 4 **Examining the Referenced Types in a Memory Snapshot**

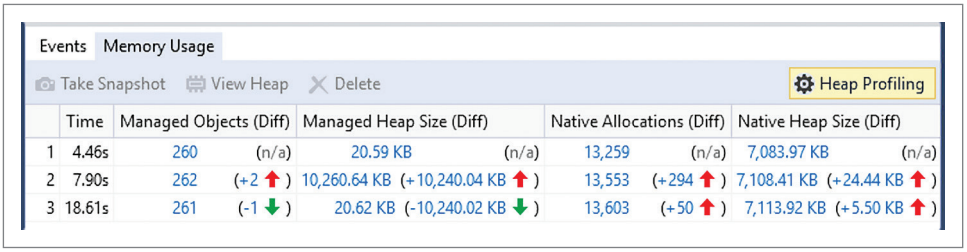


Figure 5 **Tracking Both Managed and Native Heap Usage**

allocating large objects later on. Note that the garbage collector generally does a very good job on its own without prompting from the app—you should profile your app carefully before deciding whether you need to force a collection at any time:

```
private void CleanUp(){
  buffer.Clear();
  GCSettings.LargeObjectHeapCompactionMode =
    GCLargeObjectHeapCompactionMode.CompactOnce;
  GC.Collect();
  GC.WaitForPendingFinalizers();
  GC.Collect();
  GC.WaitForPendingFinalizers();
}
```

The example in **Figure 3** shows three snapshots, taken before allocating memory, after allocating memory and then after releasing memory (using the updated version of the code). The increase and decrease in memory usage is clear from the graph.

The blue arrows show where the snapshots were taken. The gold arrow shows where a garbage collection was done. The snapshot data is listed in the Memory Usage window below, including details of the heap. The red up arrow in this list indicates where memory usage increased relative to the previous snapshot; conversely the green down arrow shows where it decreased. The delta in this case is 10MB, which matches the allocation done in the code. The Object column lists the total number of live objects on the heap—but the more useful count is the Diff. Both counts are also hyperlinks: for example, if you click the Diff count, it will expand out a detailed breakdown of the increase or decrease in objects allocated by the app at that point in time. From this list, you can select any object to get a more detailed view. For example, select the MainPage in the object window, and this will pull up a breakdown in the Referenced Types window, as shown in **Figure 4**. The size increase in this example is clearly for the 10MB array.

The Referenced Types view shows you a graph of all types your selected type is referencing. The alternative view is the Paths to Root view, which shows the opposite—the complete graph of types rooting your selected type; that is, all the object references that are keeping the selected object alive.

You can choose to focus either on managed memory allocations, native memory allocations or both. To change this, select the Project menu, then the project Properties. On the Debug tab, select the Debugger type (Managed, Native or Mixed), then turn on Heap Profiling in the Memory Usage, as shown in **Figure 5**. In some cases, as in this example, you'll see that even though the app is forcing a garbage collection and cleaning up managed allocations, the number and size of native allocations actually increases. This is a good way to track the full memory usage effects of any operation your app performs, rather than focusing solely on the managed side of things.

This is turned off by default because profiling the native heap while debugging will significantly slow down the app's performance. As always, profiling should be done on a range of target devices—and preferably using real hardware rather than emulators, as the characteristics are different. While profiling is useful during development, your app can continue to use the MemoryManager and related APIs during production, for making alternate feature decisions in production and for telemetry. On top of that, because they can be used outside the debugging environment—and without the memory overhead of debugging—they more accurately represent the app's behavior in real use.

Wrapping Up

Users typically install many apps on their devices, and many apps have both foreground and background components. Resource policy strives to ensure that the limited resources on the device are apportioned thoughtfully, in a way that matches the user's expectations. Priority is given to activities the user is immediately aware of, such as the foreground app, background audio or incoming VoIP calls. However, some resources are also allocated to less important background tasks, to ensure that, for example, the user's tiles are updated in a timely manner, e-mail is kept synced in the background, and that app data can be kept refreshed ready for the next time the user launches an app.

> Resource policy strives to ensure that the limited resources on the device are apportioned thoughtfully, in a way that matches the user's expectations.

Resource policy is consistent across all Windows 10 platforms, although it also allows for variability in device capabilities. In this way, a UWP app can be written to target Windows desktop, mobile, Xbox, HoloLens or IoT while being resilient to device variation. The app platform offers a set of APIs the app can use to track its resource usage, to respond to notifications from the system when interesting resource-related events happen and to tune its behavior accordingly. The app developer can also use debugging tools in Visual Studio to profile his app, and eliminate memory leaks. ■

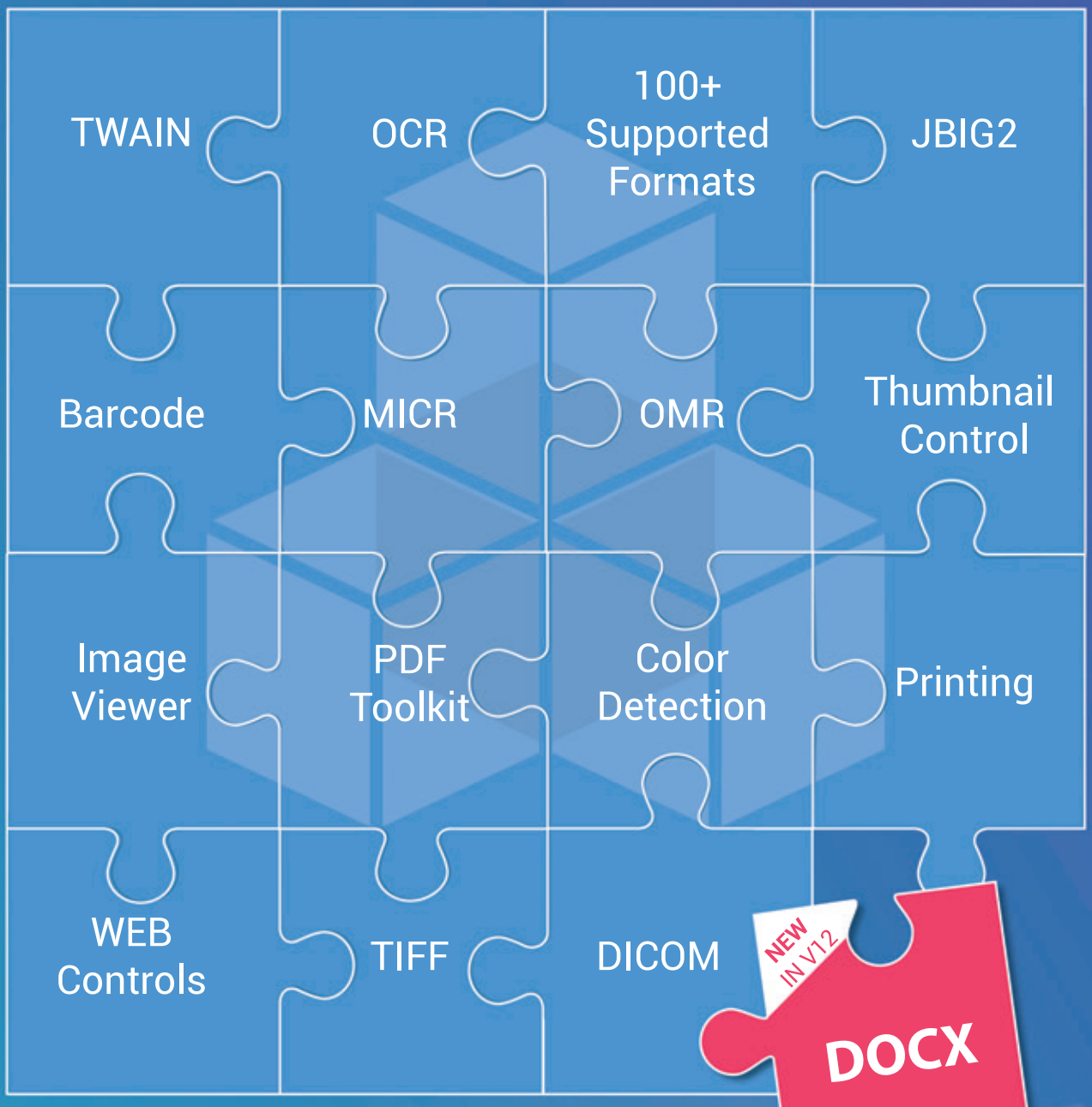
Andrew Whitechapel *is a program manager in the Microsoft Windows division, responsible for the app execution and resource policy for the Universal Windows Application Platform.*

Thanks to the following technical experts for reviewing this article: *Mark Livschitz and Jeremy Robinson*

GdPicture.NET 12

100% ROYALTY FREE

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



Try **GdPicture.NET V12** for **Free** for 30 days

www.gdpicture.com

GdPicture.NET is an ORPALIS imaging technologies product

Create Interactive Geo-Applications Using Bing Maps 8

James McCaffrey

Enterprises are generating huge amounts of data, and a lot of that data contains latitude and longitude location information. The Bing Maps version 8 library, released in June 2016, has many new features that allow you to create interactive geo-applications. In this article I present two Web applications that demonstrate some of the most interesting features of the Bing Maps 8 library. The first application highlights some of the features that allow user interaction, including a new drawing control and full event model capabilities. The second application highlights some of the features that allow users to deal with large amounts of data, including a new data clustering module and heat map visualizations.

This article discusses:

- Creating pushpins and polygons
- Initializing the map object
- Creating and displaying custom pushpins
- Retrieving interactive shapes
- Visualizing data with heat maps and clustered pushpins

Technologies discussed:

Bing Maps 8, JavaScript, HTML5

Code download available at:

msdn.com/magazine/1016magcodes

This article assumes you have basic familiarity with Web application development but doesn't assume you know anything about geolocation applications or the Bing Maps 8 library. The two demo Web applications use only standard HTML and standard JavaScript—no ASP.NET, and no JavaScript framework-of-the-month. Each of the two demo Web applications is contained in a single HTML file. The complete source code and the two data files used are available in the accompanying code download.

Take a look at the first Web application in **Figure 1**. When the Web page loaded, the HTML controls on the left were rendered immediately while the map was being fetched asynchronously. The map object is centered near Portland, Ore., and a default-style purple pushpin marker was placed at the map center.

I then clicked on the HTML5 File control Browse button and pointed to a text file named LatLonData.txt, stored on my local machine in the C:\Data directory. The file has four data points and each has some associated text. Then I clicked on the button control labeled Place Pushpins and the application read the text file, created four custom-styled small orange pushpins and placed them on the left side of the map.

Next, I clicked on the polygon item in the drawing tools control in the upper-right part of the map and interactively drew a four-sided green polygon just above the city of Vancouver, Wash. I drew a second polygon with three sides, to the right and below the map center. During drawing, the Web application listened for

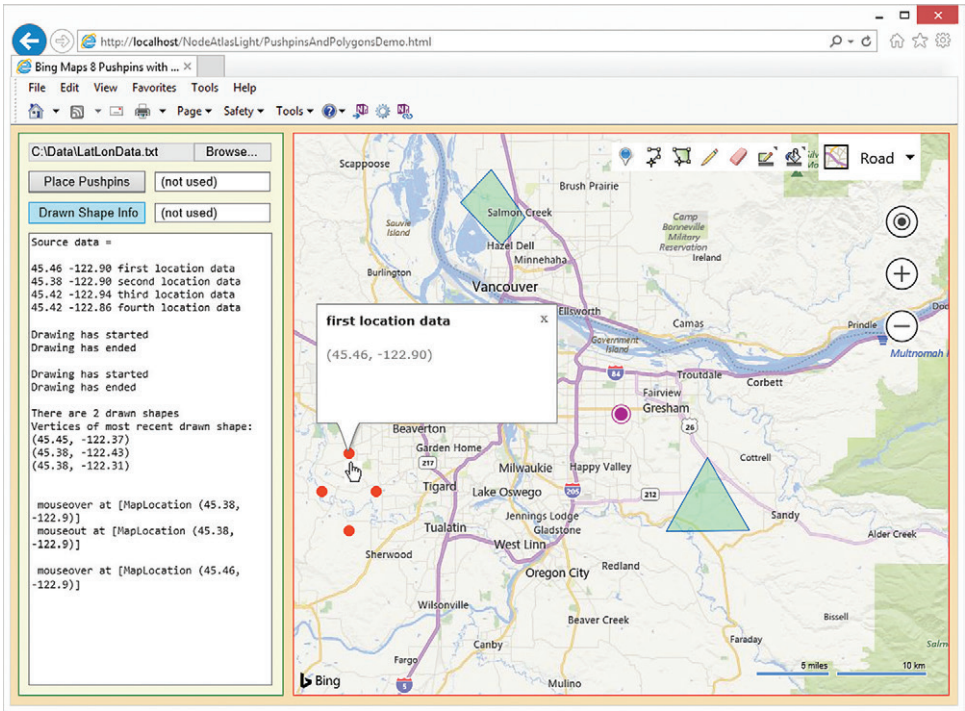


Figure 1 **Pushpins and Polygons Demo**

drawing-start and drawing-end events, and printed messages when those events fired.

I clicked on the button control labeled Drawn Shape Info and the Web application retrieved information about the interactively created polygons, and displayed the three vertices of the triangle polygon. Next, I moved my mouse cursor over and then away from the bottom-most orange pushpin. The application code caught the mouseover and mouseout and displayed the location of the events. Although it's not visible in the image, when I moved my mouse cursor over the pushpin, a popup Infobox object appeared, and when I moved the mouse cursor away, the Infobox automatically disappeared.

I finished my demo session by moving the mouse cursor over the top-most orange pushpin and the application responded by creating a default-style Infobox object that displayed data associated with the pushpin (the text "first data location") and the location of the pushpin (45.46, -122.90).

To summarize, the first Web application demonstrates asynchronous map loading, dynamic custom pushpins, rich event modeling, interactive shape creation and Infobox objects.

Creating the Pushpins and Polygons Demo Application

Before I started writing the first Web page, I created the source data file using Notepad:

```
45.46,-122.90,first location data
45.38,-122.90,second location data
45.42,-122.94,third location data
45.42,-122.86,fourth location data
```

I didn't hit the <enter> key after the last line of data so my file-reading code wouldn't try to interpret an empty line of text. I used commas as the field delimiter, but I could have used the tab character. I saved the data file as LatLonData.txt in the C:\Data directory on my local machine. As you'll see, Bing Maps can work with any kind of data store.

I used the Notepad program to create the demo Web applications. I like Notepad when learning a new technology because it forces me to be careful and there's no hidden magic to obscure the key ideas.

Because I used only plain vanilla HTML and JavaScript, I didn't need to do anything special to prepare IIS or my machine. I created a directory named NodeAtlasLight in the C:\inetpub\wwwroot directory on my machine. That name is arbitrary and you can use whatever name you like if you want to run the demo Web applications.

I launched Notepad using the "Run as administrator"option so I'd be able to save my code under the protected C:\inetpub root directory. I named the application PushpinsAndPolygonsDemo.html, but the Bing Maps 8 library has no required naming conventions, so you can use a different filename if you wish.

> It's possible to load the Bing Maps 8 library synchronously, but an asynchronous load gives a better UX in situations where the library is slow to load.

The overall structure of the Web application is shown in **Figure 2**. Here's a highly abbreviated version of the structure:

```
<html>
  <head>
    <script type='text/javascript'>
      // All JavaScript here
    </script>
  </head>
  <body>
    <!-- all HTML here -->

    <script type='text/javascript'
      src='http://www.bing.com/api/maps/mapcontrol?callback=GetMap'
      async defer></script>
  </body>
</html>
```

The key code is the <script> tag located at the bottom of the <body> section. You can loosely interpret this to mean, "Load the basic Bing Maps 8 library asynchronously while the HTML

is rendering. When the library has loaded, transfer control to a JavaScript function named GetMap." It's possible to load the Bing Maps 8 library synchronously, but an asynchronous load gives a better UX in situations where the library is slow to load.

The overall layout of the Web page consists of two side-by-side floating areas. The left-side holds the HTML controls. The right-side holds the Map object:

```
<div id='mapDiv' style="float:left; width:700px; height:600px;
  border:1px solid red;">
</div>
```

It's possible to have multiple Map objects for specialized scenarios. Instead of specifying the map width and height using pixel units, you can also use the CSS3 viewport units, vw and vh. For simplicity, I embed all HTML styling directly rather than using a separate CSS file, at the minor expense of a bit of messiness.

To summarize, a Bing Maps 8 map object is created using a program-defined JavaScript function, and is placed in an HTML area that specifies the size of the map. You can load a map synchronously or asynchronously.

Initializing the Map Object

The Web application sets up six global script-scope objects:

```
var map = null;
var pushpins = [];
var infobox = null;
var ppLayer = null;
var drawingManager = null;
var drawnShapes = null;
```

When I create a mapping application, I tend to think of the architecture as similar to a large C# or Java class, and so the script-scope JavaScript objects are typically those that are used by two or more functions. However, because of the JavaScript language's quirks and heavy use of callback functions and closures, I'll sometimes place objects that only need function-scope into the script-scope area.

> Instead of placing all visual entities into one monolithic collection, it's now possible to organize visual objects into layers.

The object named map is the Map object and although that name isn't required, it's more or less standard. The pushpins object is an array that will hold all the pushpins. I initialize the object to an empty array here, as opposed to setting it to null, mostly to indicate that the object is an array. The infobox object is a single instance of the Infobox class that will be shared by all pushpins.

One of the new features of Bing Maps 8 is the Layer class. Instead of placing all visual entities into one monolithic collection, it's now possible to organize visual objects into layers. The drawingManager object is a reference to the DrawingTools control. The drawnShapes object is an array that will hold the Polygon object shapes drawn by a user.

The map is initialized by function GetMap. The definition begins with:

```
function GetMap()
{
  var options = {
    credentials: "AmUck2_xxxx_jSCm",
    center: new Microsoft.Maps.Location(45.50, -122.50),
    mapTypeId: Microsoft.Maps.MapTypeId.road,
    zoom: 10, enableClickableLogo: false, showCopyright: false
  };
...
```

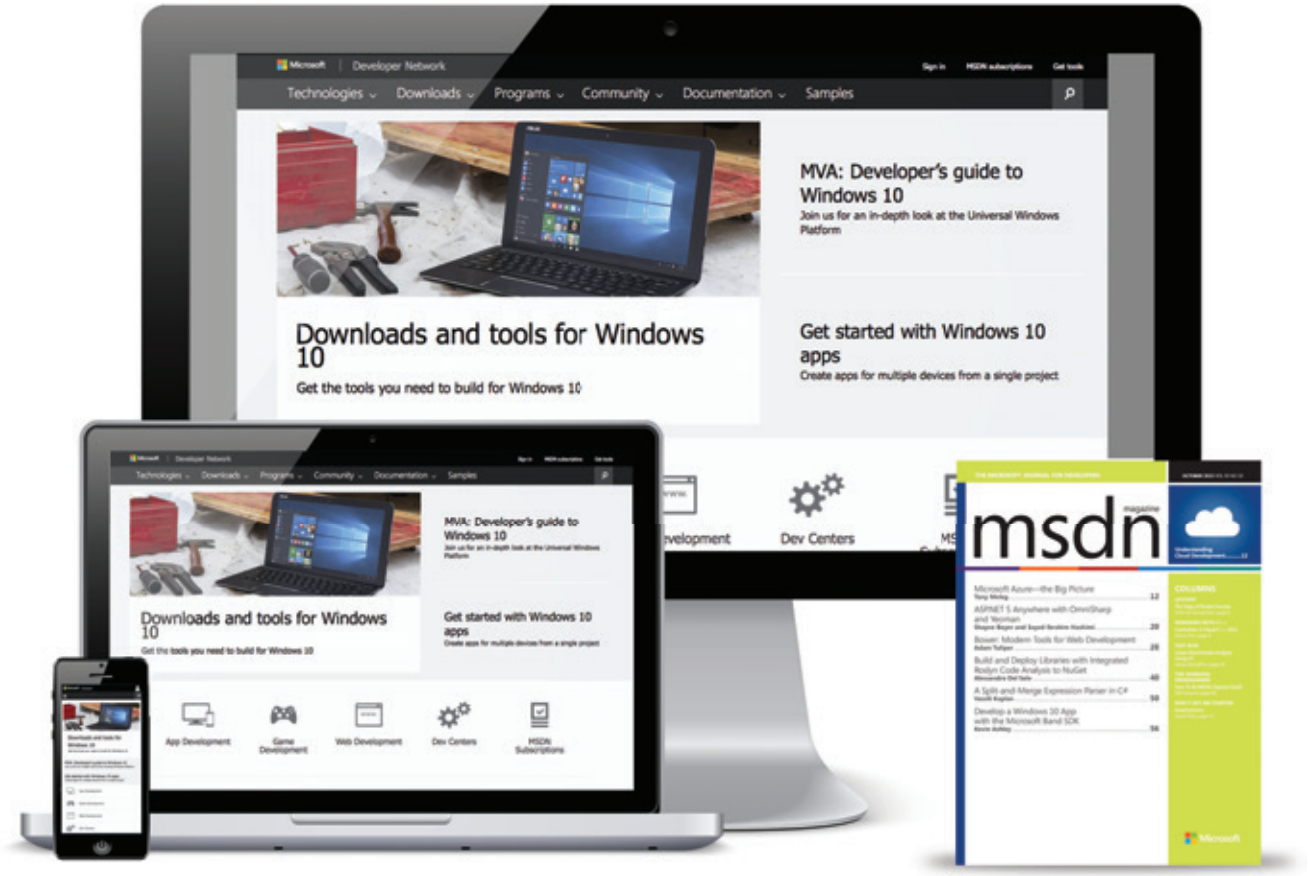
Note that I like to capitalize the names of my program-defined functions to distinguish them from library functions or built-in JavaScript functions. The code here defines some of the initial map

Figure 2 **Pushpins and Polygons Demo Web Page Structure**

```
<!DOCTYPE html>
<!-- PushpinsAndPolygonsDemo.html -->

<html>
  <head>
    <title>Bing Maps 8 Pushpins with Infoboxes</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>

    <script type="text/javascript">

    var map = null;
    var pushpins = [];
    var infobox = null;  // Shared infobox for all pushpins
    var ppLayer = null;  // Pushpin layer
    var drawingManager = null;
    var drawnShapes = null;  // an array

    function GetMap() { . . }
    function AddDrawControlEvents(manager) { . . }
    function WriteLn(txt) { . . }
    function LatLonStr(loc) { . . }
    function Button1_Click() { . . }
    function Button2_Click() { . . }
    function ShowInfobox(e) { . . }
    function HideInfobox(e) { . . }
    function CreateCvsDot(radius, clr) { . . }

    </script>
  </head>

  <body style="background-color:wheat">
    <div id='controlPanel' style="float:left; width:262px; height:580px;
      border:1px solid green; padding:10px; background-color: beige">

      <input type="file" id="file1" size="24"></input>
      <span style="display:block; height:10px"></span>

      <input id="button1" type='button' style="width:125px;"
        value='Place Pushpins' onclick="Button1_Click();"></input>
      <div style="width:2px; display:inline-block"></div>
      <input id="textbox1" type='text' size='15' value=' (not used)'></input><br/>
      <span style="display:block; height:10px"></span>

      <input id="button2" type='button' style="width:125px;"
        value='Drawn Shape Info' onclick="Button2_Click();"></input>
      <div style="width:2px; display:inline-block"></div>

      <input id="textbox2" type='text' size='15' value=' (not used)'></input><br/>
      <span style="display:block; height:10px"></span>

      <textarea id='msgArea' rows="34" cols="36"
        style="font-family:Consolas; font-size:12px"></textarea>
    </div>

    <div style="float:left; width:10px; height:600px">
    <div id='mapDiv' style="float:left; width:700px; height:600px;
      border:1px solid red;"></div>
    <br style="clear: left;" />  <!-- magic formatting -->

    <script type='text/javascript'
      src='http://www.bing.com/api/maps/mapcontrol?callback=GetMap'
      async defer></script>
  </body>
</html>
```

msdn magazine

Where you need us most.



Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

LIVE! 360
TECH EVENTS WITH PERSPECTIVE

MSDN.microsoft.com

settings. All of these are optional, even the credentials item, which is essentially a Bing Maps key. If you don't have a key, you can use any string there and your map will still load and be functional, but there'll be a thin strip across your map with a message, "The specified credentials are invalid. You can sign up for a free developer account at http://www.bingmapsportal.com." Creating an account to get a key is relatively painless, but if you're impatient like me and you want to get started right away, you can sign up later.

The map's center property is set using a Location object, which accepts a latitude, followed by a longitude, followed optionally by two values related to altitude. If you're new to geo-applications, you have to be a bit careful. With a normal geometry point (x, y) the x value is the "left-right" value, but in geo-applications the latitude is the "up-down" value.

Next, the map is displayed and the master Infobox object is prepared:

```
var mapDiv = document.getElementById("mapDiv");
map = new Microsoft.Maps.Map(mapDiv, options);
infobox = new Microsoft.Maps.Infobox(new Microsoft.Maps.Location(0, 0),
  { visible: false, offset: new Microsoft.Maps.Point(0,0) });
infobox.setMap(map);
```

One of the nice things about Bing Maps is that the API set uses variable and parameter names that are, for the most part, quite understandable. The Infobox is placed at Location (0, 0), which is just a dummy location because the visible property is set to false. The offset property controls the positioning of the small triangular pointer at the bottom of the Infobox object. The default value is (0, 0), so I could've omitted it.

Next, the pushpins are prepared:

```
ppLayer = new Microsoft.Maps.Layer();
var cpp= new Microsoft.Maps.Pushpin(map.getCenter(), null);
ppLayer.add(cpp);
map.layers.insert(ppLayer);
```

The ppLayer ("pushpin layer") object defines a visual layer where all the pushpins will be stored. The cpp ("center pushpin") is added to the Layer and then the Layer is added into the map making the pushpin visible. The second parameter to the Pushpin constructor, which is null here, can be a PushpinOptions object, which will be explained shortly. Passing a value of null gives you a default Pushpin object, which is purple and has a radius of about 10 pixels.

> A big architecture change for Bing Maps 8 is that the library is now organized into 11 modules.

Bing Maps 8 supports the older mechanism of placing all visual objects into one global collection. The code would look like:

```
map.entities.push(cpp);
```

The GetMap function finishes by creating the DrawingTools control and placing it onto the map:

```
...
    Microsoft.Maps.loadModule('Microsoft.Maps.DrawingTools', function() {
    var tools = new Microsoft.Maps.DrawingTools(map);
    tools.showDrawingManager(AddDrawControlEvents);
  });
}
```

A big architecture change for Bing Maps 8 is that the library is now organized into 11 modules. This allows you to load only those modules you need, which can significantly improve performance. Representative other modules include Search, SpatialMath and HeatMap.

The loadModule function accepts the name of the module to load, plus a callback function definition that contains code to execute after the module has loaded. It can take a while to become comfortable with callback functions, but like anything else, after a few examples you get the hang of using them.

The showDrawingManager function also accepts a callback function, this time using a name (AddDrawControlEvents) rather than an anonymous function. Function AddDrawControlEvents is defined as:

```
function AddDrawControlEvents(manager)
{
  Microsoft.Maps.Events.addHandler(manager, 'drawingStarted',
    function(e) { WriteLn('Drawing has started'); });
  Microsoft.Maps.Events.addHandler(manager, 'drawingEnded',
    function(e) { WriteLn('Drawing has ended \n'); });
  drawingManager = manager;
}
```

This code is short but rather subtle. In words, "When a user starts drawing a shape using the DrawingTools control and the drawingStarted event automatically fires, place a message using a program-defined function named WriteLn." The Events.addHandler function accepts an event-firing object and a callback function. The event argument, e, isn't used in the demo but it represents the drawn shape.

Program-defined function WriteLn is defined as:

```
function WriteLn(txt)
{
  var existing = msgArea.value;
  msgArea.value = existing + txt + "\n";
}
```

The msgArea object is an HTML textarea tag on the left side of the Web page. The approach used here of grabbing the existing content and then replacing it with appended text is rather crude but works well as long as the amount of text doesn't get huge.

Creating and Displaying Custom Pushpins

When a user clicks on the button control labeled Place Pushpins, control is passed to the Button1_Click function. The structure of the function is:

```
function Button1_Click()
{
  var f = file1.files[0];
  var reader = new FileReader();
  reader.onload = function(e) {
    // Parse each line of result
    // Create pushpins
    // Add event handlers for pushpins
    // Display pushpins
  }
  reader.readAsText(f);
}
```

Local object f has information about the physical file pointed to by the browsing control of the HTML file1 object. Because the HTML File API allows multiple files to be selected, the first file is accessed as files[0]. The FileReader object will load a file asynchronously so the Web page will remain responsive. The onload event will fire when the file has been read into memory. Notice that you define what to do after the file is read and then you call the readAsText function to actually start reading the file.

The anonymous function that executes when the onload event fires begins with:

```
WriteLn("Source data = \n");
var lines = reader.result.split('\n');
for (var i = 0; i < lines.length; ++i) {
  var line = lines[i];
...
```

The file contents are stored in the reader.result object as one giant string with embedded '\n' characters. The String.split function is used to extract each line into an array. Then the lines are iterated through using a for-loop with the length property. Next:

```
var tokens = line.split(',');
WriteLn(tokens[0] + " " + tokens[1] + " " + tokens[2]);
var loc = new Microsoft.Maps.Location(tokens[0], tokens[1]);
```

Recall that a line of the data file looks like:

```
45.46,-122.90,first location data
```

Each line is split on the comma delimiter and the three results are stored into an array named tokens, so the latitude is at tokens[0] and the longitude is at tokens[1]. Because a lot can go wrong when reading a text file, in a production system you'd likely wrap the attempt to create a Location object in a JavaScript try-catch block.

Next, a custom pushpin is created for the data of the current line of text:

```
var ppOptions = { icon: CreateCvsDot(6, "orangered"),
  anchor: new Microsoft.Maps.Point(6,6), subTitle: tokens[2] };
var pp = new Microsoft.Maps.Pushpin(loc, ppOptions);
pushpins[i] = pp;
```

Custom pushpins are created by passing information to the icon property of a PushpinOptions object. Here, a custom orange-red color icon with a radius of 6 pixels is created by calling a program-defined function named CreateCvsDot. I also set the subTitle property of the current pushpin to the text from the data file that follows the lat-lon fields. After the pushpin is created, it's added to the global pushpins array.

The anonymous function code finishes with:

```
  ...
  Microsoft.Maps.Events.addHandler(pushpins[i], 'mouseover', ShowInfobox);
  Microsoft.Maps.Events.addHandler(pushpins[i], 'mouseout', HideInfobox);
}
ppLayer.add(pushpins);
map.layers.insert(ppLayer);
WriteLn("");
```

Each pushpin has its mouseover and mouseout events modified using program-defined functions ShowInfobox and HideInfobox. After all pushpins have been created, the array holding them is adding to the pushpin Layer, which is then inserted into the map, which makes the pushpins visible.

Function CreateCvsDot ("create HTML canvas dot") is defined as:

```
function CreateCvsDot(radius, clr)  {
  var c = document.createElement('canvas');
  c.width = 2 * radius; c.height = 2 * radius;
  var ctx = c.getContext("2d");
  ctx.beginPath();
  ctx.arc(radius, radius, radius, 0, 2 * Math.PI);
  ctx.fillStyle = clr; ctx.fill();
  return(c.toDataURL());
}
```

The function accepts a radius and a color and returns an HTML5 canvas object. There are four ways to create a custom pushpin icon. You can use a static image such as a .png file; you can use a static image encoded using Base64 format; you can create a dynamic HTML canvas object; or you can create a dynamic scalable vector graphics (SVG) object.

The ability to create a pushpin icon on the fly gives you a lot of flexibility. For example, you could create different color and size icons depending on the density of pushpins in an area of your map, or depending on the zoom level of the map.

Event-handler function ShowInfobox is defined as:

```
function ShowInfobox(e)
{
  var loc = e.target.getLocation();
  WriteLn('\n mouseover at ' + loc);
  infobox.setLocation(loc);
  infobox.setOptions( { visible: true, title: e.target.getSubTitle(),
    description: LatLonStr(loc) });
}
```

When the user moves the mouse cursor over a pushpin, the pushpin's mouseover event will fire and control will transfer to ShowInfobox. The function gets the Location of the event/pushpin and uses it to place the pushpin. Recall that the subTitle property of each pushpin holds text such as "first data location." This text is used as the Infobox title.

> Custom pushpins are created by passing information to the icon property of a PushpinOptions object.

The description property of the Infobox is set to the location of the pushpin, formatted to two decimal places using the program-defined helper function LatLonStr:

```
function LatLonStr(loc)
{
  var s = "(" + Number(loc.latitude).toFixed(2) + ", " +
    Number(loc.longitude).toFixed(2) + ")";
  return s;
}
```

The HideInfobox function is:

```
function HideInfobox(e)
{
  WriteLn(' mouseout at ' + e.target.getLocation());
  infobox.setOptions({ visible: false });
}
```

When the user moves the mouse cursor away from a pushpin, the pushpin's mouseout event will fire and control will transfer to HideInfobox. The visible property is set to false so the Infobox isn't visible but is still in the map.

Retrieving Interactive Shapes

When a user clicks on the button control labeled Drawn Shapes Info, control is transferred to the Button2_Click function. The function is defined as:

```
function Button2_Click()
{
  drawnShapes = drawingManager.getPrimitives();
  var numShapes = drawnShapes.length;
  var mostRecent = drawnShapes[numShapes-1];  // Polygon
  var vertices = mostRecent.getLocations();
  WriteLn("There are " + numShapes + " drawn shapes");
  WriteLn("Vertices of most recent drawn shape: ");
  for (var i = 0; i < vertices.length; ++i) {
    WriteLn(LatLonStr(vertices[i]));
  }
}
```

The global drawingManager object was created when the DrawingTools control was placed on the map. It's used to fetch an array containing all shapes drawn by the drawing control. The last shape drawn will be the last item in the array. The code assumes the drawn

shapes are type Polygon, but the DrawingTool control can create different types of objects. You could check the shape type with code like:

```
var isPoly = mostRecent instanceof
Microsoft.Maps.Polygon;
```

The function finishes by fetching the vertices of the last drawn shape using the getLocations function, and iterating through the vertices to display them.

The Heat Map Demo

When I work with geo-applications, I mentally categorize them according to the number of data points with which I'm dealing. Working with a large number of locations can be challenging. The Bing Maps 8 library has two very nice ways to work with a large number of locations—clustered pushpins and heat maps. Take a look at the demo heat map in **Figure 3**.

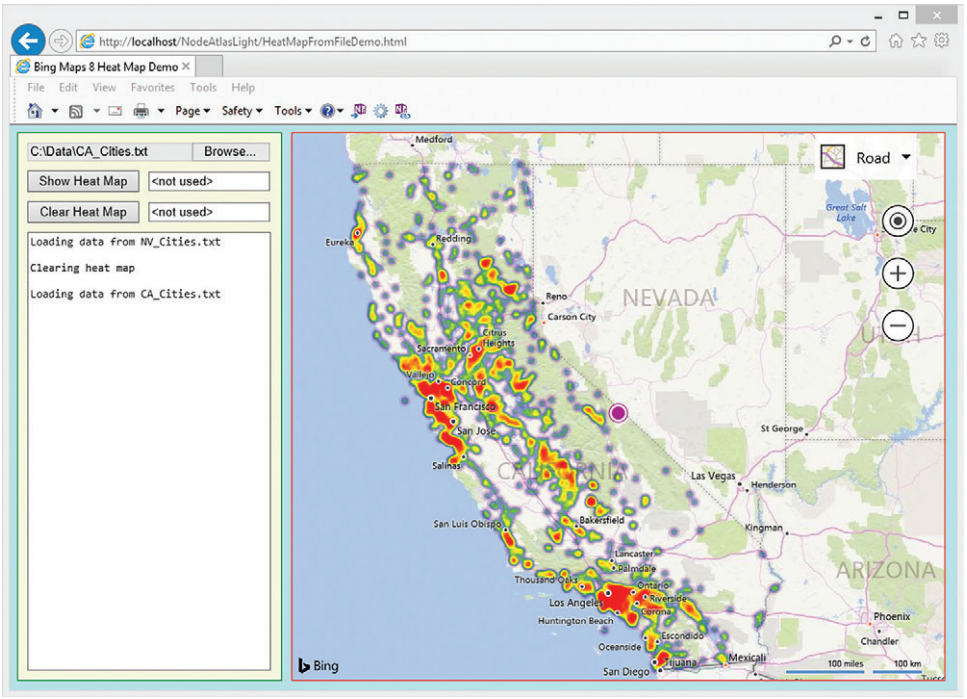


Figure 3 **Heat Map Demo**

There are several kinds of heat maps, but one common type displays combined data points using a color gradient where different colors represent different data densities. The demo Web application initially loads a map centered at (37.50, -118.00) and places a default large purple pushpin at center.

First, I clicked on the HTML5 File Browse button and pointed to a local file named NV_Cities.txt containing city data. Next, I clicked on the first button control, which loaded and displayed a heat map for city density in the state of Nevada. Then I cleared that heat map using the second button control.

Next, I clicked on the Browse button control again and pointed to a tab-separated text file named CA_Cities.txt. That data file contains a list of 1,522 cities in California and their corresponding latitude-longitude information. Then I clicked on the Show Heat Map button control, which read the text file, parsed out the lat-lon data and stored that data into an array. The lat-lon data was then displayed as a heat map, generating a city density visualization.

The structure of the heat map demo application is almost exactly like the structure of the pushpins and polygons demo. The global script-scope objects are:

```
var map = null;
var ppLayer = null;
var hmLayer = null;
var reader = null;   // FileReader object
var locs = [];
var cGrad = { '0.0': 'black', '0.2': 'purple', '0.4': 'blue', '0.6': 'green',
  '0.8': 'yellow', '0.9': 'orange', '1.0': 'red' };
var hmOptions = { intensity: 0.65, radius: 7, colorGradient: cGrad };
```

The hmLayer object is a Layer for the heat map. The locs array holds the Location objects that define the heat map. The cGrad object defines a custom color gradient for the heat map options. The hmOptions define the options for the heat map. Using a custom HeatMapOptions object is optional, but in most situations you'll want to use the options to control the appearance of your heat map.

Here's the code in function Button1_Click that reads and parses the source data file:

```
var lines = reader.result.split('\n');
for (var i = 0; i < lines.length; ++i) {  // Each line
  var line = lines[i];
  var tokens = line.split('\t');  // Split on tabs
  var loc = new Microsoft.Maps.Location(tokens[12], tokens[13]);
  locs[i] = loc;
}
```

The source data files look like:

```
CA  602000  2409704  Anaheim city    ( . . )  33.855497  -117.760071
CA  602028  2628706  Anchor Bay CDP  ( . . )  38.812653  -123.570267
...
```

Each line has 14 tab-delimited values. The first value is the state abbreviation. The next two fields are IDs. The fourth field is the place name, which can be a city, a town or a census-designated place (CDP). Then there are eight fields that include information such as U.S. census population count and land area. The last two fields are the latitude and longitude. I got the data from the U.S. Census Web site at bit.ly/29SETIU.

The code that creates and displays the heat map is:

```
Microsoft.Maps.loadModule('Microsoft.Maps.HeatMap', function() {
  hmLayer = new Microsoft.Maps.HeatMapLayer(locs, hmOptions);
  map.layers.insert(hmLayer);
});
```

The hmLayer Layer is created using the global array of Location objects and the hmOptions object that contains the custom color gradient. Very nice!

The code for function Button2_Click removes the current heat map:

```
function Button2_Click()
{
  WriteLn('Clearing heat map' + "\n");
  hmLayer.clear();
  reader = null;
  locs = [];
}
```

This code illustrates one of the advantages of working with Layer objects. Instead of having to iterate through every object

in the Map.entities collection, you can directly access objects in a particular layer.

Pushpin Clustering

One of my favorite new features in Bing Maps 8 is pushpin clustering. The idea is best explained visually. In **Figure 4**, the Web page named ClusteredPushpinsDemo.html loads a map with an initial zoom level of 10, centered near Portland, Ore. When I clicked on the button control labeled Generate Pins, the application used the getLocations function in the Maps.TestDataGenerator to create 6,000 random locations. Then the application code created clustered pushpins and displayed them. Red circles indicate there are 100 or more pushpins in the associated map area, and blue pushpins indicate 10 to 99 pushpins.

Next, I zoomed in three levels. Clustering automatically occurs at each zoom change. At zoom level 13 (see **Figure 5**) the individual pushpins become visible as small red dots, and green circles indicate there are two to nine pushpins at that location.

Both heat maps and pushpins clustering enable you to manage a large number of location items. But using pushpin clustering allows users to access individual items.

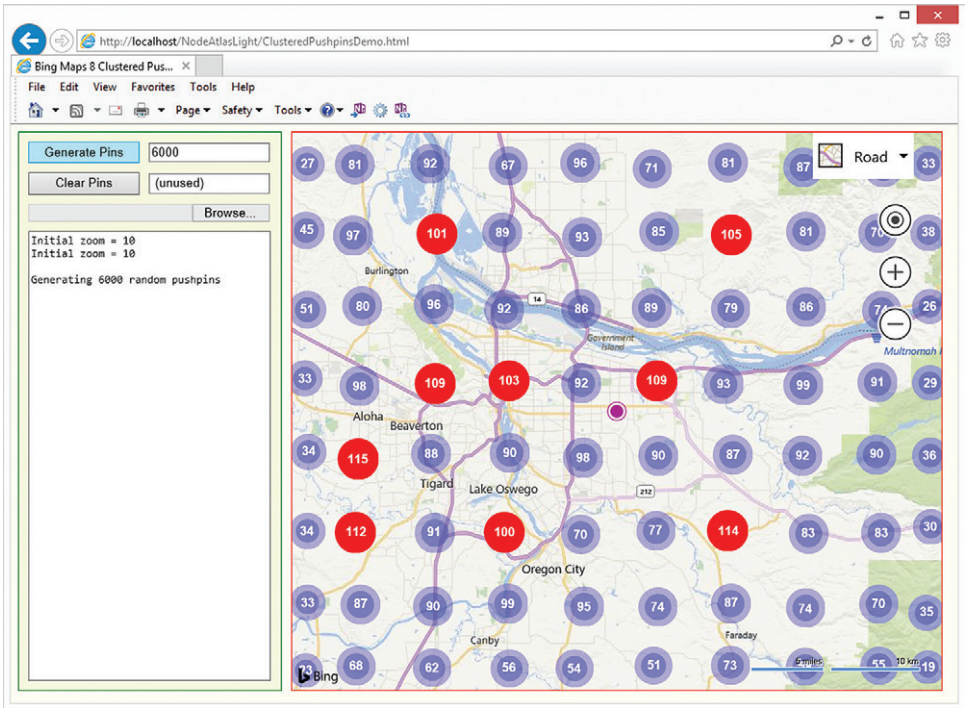


Figure 4 **Pushpin Clustering with Zoom Level 10**

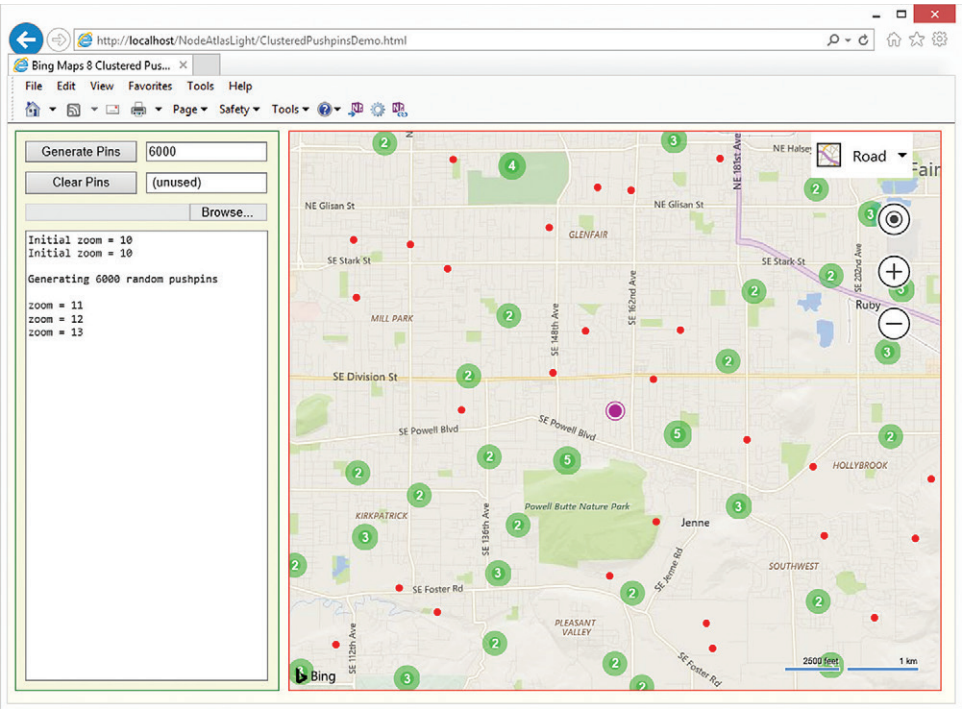


Figure 5 **Pushpin Clustering with Zoom Level 13**

Wrapping Up

The demo Web applications presented here should give you a good idea of what the new Big Maps 8 library is like. There are many additional new features that I didn't cover, including Infobox customization, tile layers, geo-search and spatial math functions. If you want to learn more about Bing Maps 8, I recommend going to the interactive SDK Web site at binged.it/29SFytX. It presents approximately 137 very short but complete Web pages that illustrate many key features of Bing Maps 8. You'll also find that the official documentation at aka.ms/BingMapsV8Docs is very well-written and useful.

I've used two of the main alternatives to the Bing Maps 8 library, the Google Maps library, and the open source Leaflet.js library. All three libraries are excellent, but I really like Bing Maps 8. Some technologies just have a "right feel" to them and for me, at least, Bing Maps 8 is now my preferred library for geo-applications. ■

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

Thanks to the following Microsoft technical experts who reviewed this article: *Ricky Brundritt (Bing Maps) and John Krumm (Microsoft Research)*

Transform Source Code to Deployable Artifacts with TFBuild

Kraig Brockschmidt

In my last article, "The Source of Truth: The Role of Repositories in DevOps" (bit.ly/2bKeC2T), I discussed the vital role that source control plays in the overall release pipeline, shown in **Figure 1**. I've described the release pipeline as the collection of processes that transform code in the source repository into customer-ready apps and services, and delivers those to customer devices and customer-accessible servers. The release pipeline, too, is simply a list of the steps that are necessary to make a release happen, irrespective of automation. The practice of DevOps thus begins with knowing what steps and processes are involved in a release, after which you can then incrementally automate those processes to lower costs and increase quality.

This article focuses on the Build/Continuous Integration (CI) stage of the pipeline for mobile apps (outlined in **Figure 1**). Positioned where it is, I think of Build this way: Build is what transforms source code into testable and deployable artifacts as needed by the rest of the release pipeline. That is, the outputs of Build are the inputs for the remaining stages of the pipeline that work or act on build artifacts. For example, all forms of testing operate against executable app and service code, not raw source code. And, of course, you must have the appropriate binaries to deploy to app stores and Web servers. Even in cases where source files go through the pipeline more or less unmodified (as with many JavaScript, HTML and CSS files in an Apache Cordova app, for instance), Build still fulfills the role of creating the appropriate packages, combining files, applying preprocessors and minifiers, and so on.

Think of Build as you would about physical construction. A large pile of building materials, such as lumber, concrete and rebar, nails, screws, windows, pipe, wire, roofing, insulation, fixtures, and so forth, contains the *potential* for a house, but the pile will not automatically turn itself into such a structure. That happens only by having someone apply the knowledge of how all those pieces get put together, step by step. That's what "doing a build" is all about. In the realm of mobile apps, especially, it might even be said that there are multiple piles of source materials, some of which are shared among target platforms, and others of which are unique.

Of course, a big advantage with software is that doing a build doesn't consume the source materials in the process. You can do a build as often as you want, as many times as you want and, if the process is automated, at very little cost. Continuous integration relies on these characteristics. You can build multiple, independent artifacts from the same source code, each of which has a separate release pipeline. This is often the case with mobile apps. In the MyDriving project (aka.ms/iotsampleapp), the example I've been referring to in this series, there are separate builds for iOS, Android and Windows, and for the code that's deployed to Azure App Service, as illustrated in **Figure 2**. (Note that you can use Build/CI for ongoing testing for your dev team without necessarily feeding into a release pipeline.)

Know, too, that Visual Studio Team Services can draw from repositories outside of a Team Project, allowing you to manage certain libraries for your app in public, open source repositories, while the proprietary portions are kept private. Any given build

This article discusses:

- The role of Build in DevOps to validate a source repository
- The advantage of continuous integration to closely associate build results with code commits
- Team Foundation Build servers, queues, agents and pools
- Setting up automated builds in Visual Studio Team Services
- Deploying a build agent to a Mac OS X machine to support iOS builds

Technologies discussed:

Visual Studio, Visual Studio Team Services, Team Foundation Server, Team Foundation Build, Build Definitions, Build Agents, Agent Pools, Continuous Integration

definition can draw from only a single repository, but a Team Project can employ any number of build definitions.

The Role of Build in DevOps

Within professional construction crews that work on houses and other buildings, someone is always checking whether all the necessary materials are on hand for at least the next few days of work. This continuous validation improves the efficiency and productivity of the crew, which is to say, their performance, and is essential for delivering results on time and within budget. Build accomplishes the same thing in software. As I said in "From Code to Customer: Exploring Mobile DevOps" (bit.ly/2ayD9Zw), all DevOps activities and practices are means to continually validate the performance of your apps and services. (Again, "performance" means delivering the greatest value to your customers at the lowest cost to your business.) Thus, Build is fundamentally a means to *validate* the contents of a source code repository, because you *expect* to have everything in place.

A build (excluding additional tests that might be run) generally has only one of two results: success or failure. Success means that the source repository contains the necessary, buildable files to produce testable and deployable artifacts. Failure means that one or more files are faulty (they have syntax errors), or that something is missing from the repository according to the way the build is configured. (That configuration could be itself faulty, of course.)

Ideally, you want to know as quickly as possible when a defect that "breaks the build" is introduced into the repository. This is where build automation becomes a huge advantage, because you can run a build and get immediate validation whenever there's a change to the repository. This is known as Continuous Integration.

Continuous Integration

In times past, builds were often complex, tedious processes that for large projects typically required one or more full-time dedicated employees. For this reason, they were run only infrequently, by which time many hundreds or even thousands of changes might have been committed to the repository. Because any number of those changes, in any combination, might cause the build to fail, it could be a real nightmare to get everyone's changes integrated into a working build. I even remember a few ambitious projects at Microsoft that were canceled simply because they couldn't actually be built.

Avoiding such nightmares has given rise to CI. Here's how I think of it:

1. Because Build validates your repository, you naturally want to run builds early and often.
2. If you can automate builds, you make the process highly repeatable at very little cost.
3. If you can automatically trigger a build whenever there's a change to the source repository, you've achieved CI.

CI, in short, validates each and every change to the source repository as close in time as possible to the change itself, and immediately notifies the appropriate developer if the build fails. Features like gated check-in (with Team Foundation Version Control) or pull request (PR) builder with Git can also trigger a build with the new code *before* it's checked in or merged, so that the repository is changed only if the build is successful. Either way, CI quickly detects faulty code in the repository (including code that fails automatically triggered tests, but that's the subject of a later article).

All this is why build automation with CI is one of the most common DevOps practices. Indeed, even if you do everything else in your release pipeline manually, you'll find that investing early in source control with automated builds and CI is well worth it, especially as a project becomes more complex.

The Anatomy of Builds

At minimum, Build requires three components: the source code, a build agent and a build definition, which is to say: the code you want to build; a machine that has the necessary tools and SDKs to produce artifacts from that code; and a set of instructions that tell the machine how to go about it. The basic relationships are illustrated in **Figure 3**. (Again, it's certainly possible to have multiple repositories, build definitions and build agents, as suggested in **Figure 2**.)

Even if the terms "build agent" and "build definition" seem new, you've actually been using them since your very first day of coding with a quintessential "Hello, World!" program:

1. By installing some programming tools on your machine, you turned it into a build agent.
2. By writing a short program and storing it in a file, you created your first piece of source code.
3. By typing in a command to compile and link that code, you created a build definition and ran a build, resulting in a runnable executable. (Modern JIT tools like Microsoft .NET Core typically compile and run the program together.)

Of course, as soon as you got a taste for the magic of coding you started writing a lot more code, factoring that code into multiple files and creating much more complex projects. At that point, typing in commands over and over became

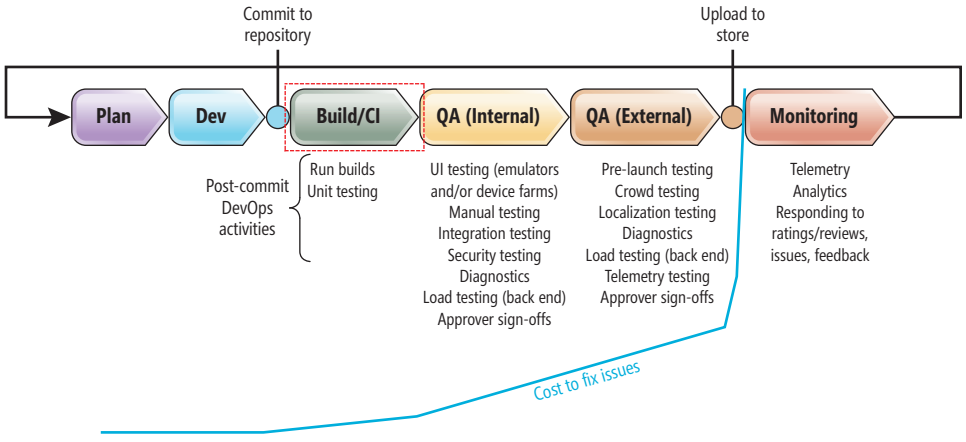


Figure 1 **Build Transforms Source Code into the Artifacts Needed in the Rest of the Release Pipeline**

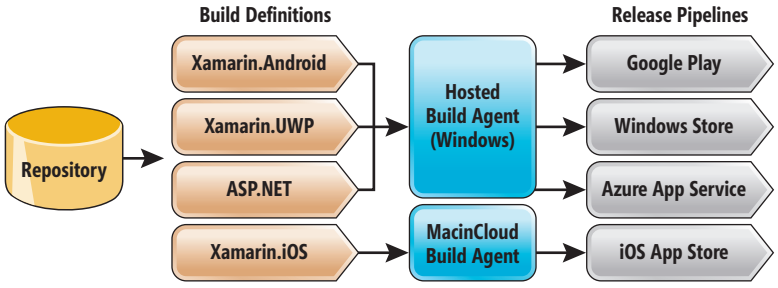


Figure 2 **The MyDriving Project Has Four Builds and Four Release Pipelines**

tedious, so your build definitions likely took the form of batch files or other scripts. Eventually you also got tired of waiting for everything to recompile every time you ran a build and, thus, discovered the virtues of systems such as NMAKE and MSBuild with makefiles and project files, respectively, serving as build definitions. These systems let you define the interrelationships between files such that you recompile only what's necessary, greatly shortening your edit-build-test dev loop.

All of this is to say that as software development continues to evolve, so does the sophistication of the build tools you have at your disposal. Most recently, making these tools available as scalable cloud services—what we call Team Foundation Build or TFBuild—has laid the groundwork for the next generation.

Servers, Queues, Agents and Pools

When working by yourself, you'll gradually install more tools and SDKs on your development machine, making it a richer and more capable agent. Before long, though, it makes sense to create one or more dedicated build servers on which you can cleanly manage the software environment. This is especially important with teams, because it avoids having to keep every developer machine in sync with the necessary tools. Managing such build servers, along with other collaborative tasks like source control, work tracking, and testing, drove the creation of the Microsoft Team Foundation Server (TFS) and TFBuild more than a decade ago.

An important feature of TFBuild, especially with CI, is its ability to manage and coordinate multiple build requests through a queue. If you have many developers committing code throughout the day, many commits will undoubtedly happen when another build is already in progress. But you don't want to cancel that build, because with CI you want every succeed/fail build report to be associated with a specific commit. At the same time, the more requests that get stacked up in the queue, the longer developers have to wait for build results.

At this point you need to scale the system, meaning that a build agent becomes distinct from a machine. Technically speaking, a build agent is a service: Multiple agents can run on the same physical server, allowing full utilization of multi-core machines by running parallel builds. And when that server is maxed out, you can easily add one or more machines with additional agents.

This creates what's called an agent pool, an abstraction that refers to the combined power of all the agents, regardless of how they're distributed across physical machines. TFBuild, in fact, works with agent pools rather than machines. When a build gets to the front of the queue, TFBuild delegates it to the next available agent in the pool. Agent pools can also be shared across different Team Projects, as explained on the "Administer Your Build and Deployment System" page at bit.ly/2bOUwAg.

With mobile app projects, especially, you'll typically need more than one kind of agent because each target system has a unique set of platform-specific tools and SDKs. Fortunately, Microsoft provides free agents for Windows, OS X, and Linux, as illustrated in **Figure 4**. Later in this article I'll demonstrate setting up a Mac agent for iOS builds.

Build Servers in the Cloud

Any conversation about scaling naturally brings up the potential for migrating that computing power to the cloud, where policies and regulations allow. With cloud-based servers, you don't need to manage the physical machines or even the software environment. Cloud computing is also centered on pay-as-you-go pricing, making it easy and cost-effective to vary your capacity as needed. This is what's offered through Visual Studio Team Services.

Team Services provides "hosted" build agents. These are Windows machines that are pre-configured with a wide variety of tools and SDKs, as listed on bit.ly/2aNFKis. As of this writing, a hosted agent can build just about anything you can build with Visual Studio and the Java toolchain, including Windows and Android apps written with Xamarin, Apache Cordova, or native tools. You can set up any number of hosted agent pools and organize them as desired into different pricing tiers.

Because the hosted agents run on Windows, however, they can't build iOS apps or .NET Core/ASP.NET Core apps for Mac or Linux. For these you need to install the OS X or Linux agents on suitable machines and connect those agents to Team Services, where you can then organize them into pools. Similarly, you can install the Windows agent on your own customized machine that includes software not included on the hosted agents. And "machine" here includes virtual machines, as well as those from services like MacinCloud.com. All in all, Team Services really lets you work with any combination of cloud-hosted and on-premises agents.

Automating Builds with Visual Studio Team Services

Let's now set up an automated TFBuild with continuous integration for a Xamarin app (roughly following "Build Your Xamarin App" at bit.ly/2aiy48y). To begin, create a new Xamarin.Forms project in Visual Studio called Xamarin Build Oct 2016. Next, create a new Team Project for it in your Team Services account called MSDN Magazine Oct 2016, using Git for source control. Then publish the code into the Team Project from within Visual Studio Team Explorer. This results in the code being available in the Team Project code tab on the Team Services portal.

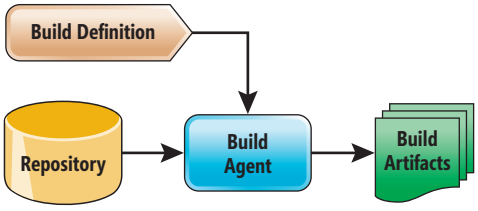


Figure 3 **The Basic Relationships Between a Source Repository, a Build Definition, a Build Agent and the Resulting Build Artifacts**

LightningChart redefines .NET charting performance standards with

1 BILLION

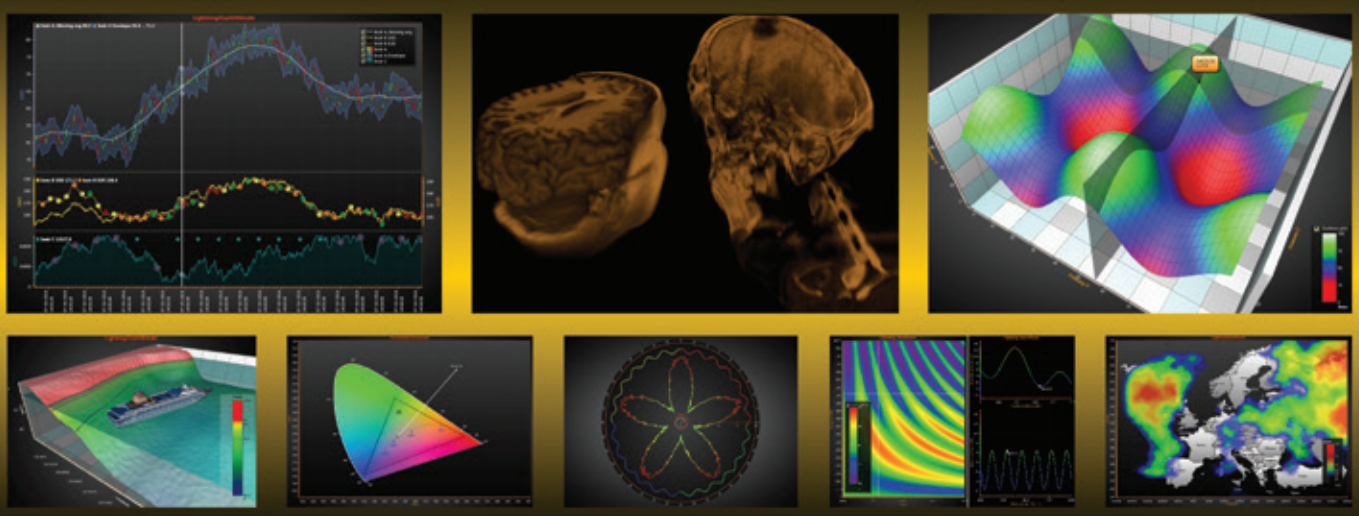
data points real-time charting

1,000,000
data points?
So yesterday.

1,000,000,000
data points?
Yes, today!

Scrolling line plot, 10 x 100,000,000 points, 4K Ultra HD resolution, 2 px line, coloring with alert levels.
Avg. refresh rate > 60 FPS, using NVidia GTX 960 and Intel i7 hardware. No down-sampling, no tricks.

LIGHTNING-FAST CHARTING COMPONENTS FOR SCIENCE, ENGINEERING AND TRADING



WPF - Windows Forms

- Entirely DirectX GPU accelerated
- Includes 100's of code examples
- Optimized for real-time data monitoring
- Supports gigantic data sets
- Interactive, zoomable charts
- On-line and off-line maps
- Total configurability
- Outstanding customer support

Arction

2D charts - 3D charts - Maps - Volume rendering - Free Gauges

www.LightningChart.com

FREE TRIAL

Now set up a build definition for Android by clicking on the Build tab in the Team Project, then clicking + New. This brings up a dialog with a long list of build definition templates for a variety of self-explanatory project types, including native and cross-platform mobile apps. (Note that Apache Cordova projects need an extension from the Team Services marketplace; see bit.ly/2atxNgp for more information.) You can also start with an empty definition and build it up step-by-step.

For this walk-through, select the Xamarin.Android template, click Next to bring up a configuration dialog, and then provide initial settings for the source repository, CI, and the agent queue (all of these can be changed later). Then click Create, and Team Services opens the build definition editor shown in **Figure 5**. Red text indicates that additional information is required, as you can see with the Build tab.

Before going through the build steps shown in **Figure 5**, let me summarize what's on the other tabs (you'll find the full documentation at bit.ly/2ayghJh):

- Repository is where you connect to repositories outside of your Team Project, such as GitHub, Subversion or any other Git server.
- Triggers set when builds happen, with options for CI, gated check-in and scheduled builds (often used for nightly runs). Note that you can always queue a build manually from Team Services or from within Visual Studio Team Explorer.
- Options | Create Work Item on Failure is how you assign work items to whomever requested a build that fails. When used with CI, the requestor will always be the developer who committed the code that triggered the build, thus making a tight loop between code commits, builds and immediate notifications of failures. (There are also extensions in the Team Services marketplace for sending other types of notifications.)
- Variables let you associate tokens with optionally encrypted values that you can use elsewhere in the build definition, such as credentials. Encrypted values can't be copied out of the build definition.
- History gives you the change log for the build definition. This is important because the build definition isn't part of your source repository, and yet changes to the definition can break the build.

Returning to **Figure 5**, remember that Build is all about turning source code into the artifacts needed by the rest of the release pipeline, which means applying specific tools to the source code through discrete steps. As you can see, the first step in the Xamarin.Android template is NuGet restore, because you typically don't add such packages to source control. When the build agent retrieves that code for a build, then, it must restore those packages.

Next, the template includes (as of this writing) steps to activate and deactivate a Xamarin license, which are no longer necessary because Xamarin is now free. It's safe to delete those steps because they'll be removed from the template soon enough anyway.

The Build Xamarin.Android Project step is what runs MSBuild on the Android project in the solution. Clicking that step shows the options as in **Figure 5**, where the More Information link at the bottom takes you to the detailed documentation for the step. Notice the $( ) references to variables, such as $(BuildConfiguration), which is set to "Release" on the Variables tab. Also, Output Directory is where the build definition will place its artifacts for use by other build steps and even other parts of the release pipeline.

The next two steps build any project in the solution that contains the name "test" and then deploys the built app to Xamarin Test Cloud and runs applicable tests. This, along with other test steps added through the + Add build step… command, is how you include test runs in your CI. Because there are no test projects in my solution at this point, I just disable these steps by clearing their Control Options | Enabled checkbox. This way the steps remain in the definition, but won't be run.

The last two steps, as you can see, sign the Android app package and then publish the artifacts to some other location, if needed. In the latter case, the earlier MSBuild step already stores its results in a Team Services folder, but that's available only to people with Team Project permissions. A publish or file copy step (there are a number of options in Add new step…) is how you copy artifacts to locations outside of the Team Project.

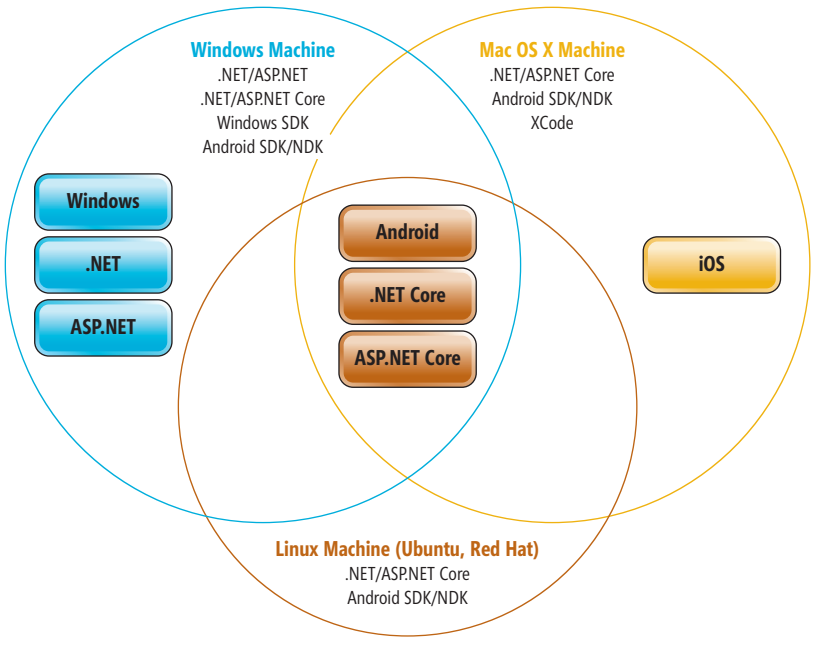


Figure 4 **Windows, Mac, and Linux Agents and the Project Types They Can Build**

Running the Build

Once you've completed and saved your build definition, click Queue build… to start the process running. This pops up a dialog where you can select the agent queue and the Git branch (if applicable), and even set additional variables and demands for just this one build. Being able to change these parameters lets you easily run a build through a different agent pool without editing the build definition. For example, if you're in the process of migrating from on-premises agents to hosted agents, you'll want to manually queue builds to the hosted queue while CI-triggered builds continue to happen on your local agents. When you're ready to switch over, you then edit the build definition and redirect the CI builds to the hosted agents instead.

In my case, I don't need to change anything, so I just start the build. This switches to a UI where I

Extreme Performance Linear Scalability

For .NET Applications

Open Source (Apache 2.0)

NCache™

Distributed Cache

- In-Memory App Data Caching
- ASP.NET Sessions & View State
- Runtime Data Sharing

NosDB

NoSQL Database

- Schema-less JSON Documents
- Data Consistency & Replication
- Powerful SQL Support

100% Native .NET

Microsoft Azure

Visual Studio

Windows Server 2012

sales@alachisoft.com www.alachisoft.com US: +1 (925) 236 3830

can see what steps have been completed, what step is running and the console output directly from the build agent. I have to admit, it's always interesting to watch a build as it progresses—I'm sure you've done that many times! And once the build is done you can download the output for further analysis if needed.

Eventually the build will complete successfully or encounter a failure. Either way, you'll see a final report, which appears in the list of Completed builds for the definition, as shown in the left side of **Figure 6**. This is where you can go back and review however many builds are being retained in your Team Project and, of course, queue new builds. This same list appears in the Visual Studio Team Explorer pane, as shown in the right side of **Figure 6**. From here you can also queue new builds and create new build definitions.

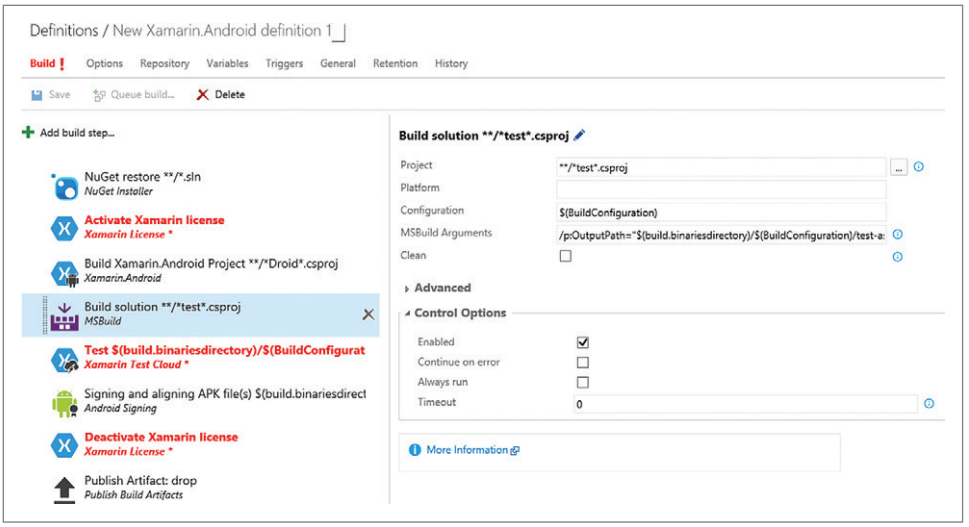


Figure 5 **The Initial View of a New Xamarin.Android Build Definition**

I encourage you to play around with this yourself using a new project created from a template in Visual Studio. Especially try checking Continuous Integration in the definition's Triggers tab, then make some small change in a source file code and commit/push it to the repository. You can then see the new build queued automatically, which appears in the definition's Queued list in Team Services and in Visual Studio Team Explorer.

Additional TFBuild Steps

A build definition template is obviously just a place to start, and you can just as easily start with an empty definition and add each step individually. Either way, you want to fully familiarize yourself with all of the available steps by opening any build definition and clicking on the Add build step… command. From the Add tasks dialog that appears, you can build a wide variety of custom build definitions. Here's an overview of what's available (the current list is always at bit.ly/2biafxz):

- **Build tasks:** Android (including signing), Ant, CMake, Gradle, Grunt, Gulp, queue a job on a Jenkis server, Maven, MSBuild, SonarQube analysis for MSBuild, Visual Studio Build, Xamarin.Android, Xamarin.iOS, Xcode (build and package).
- **Test tasks:** Cloud-based load testing with Apache JMeter or Visual Studio Team Services; cloud-based performance testing; publish code coverage and test results; run functional tests (CodedUI or Selenium); Visual Studio Test (including deploying agents); Xamarin Test Cloud.
- **Package tasks:** Run NuGet and npm tasks, restore Xamarin components, install CocoaPods.
- **Deploy tasks:** Deploy to Azure Cloud Service, blob, VM, SQL database, WebApp; run Azure PowerShell scripts; run PowerShell or SSH scripts on target machines; manage Azure Resource Groups; deploy Azure Service Fabric applications; copy files to a remote machine.
- **Utility tasks:** manage zip files, copy/decrypt/delete files; run batch files or command-line tools (including bash and PowerShell); upload files with cURL; and copy files.

There's also a healthy marketplace of additional tasks to choose from at bit.ly/2aiRnPh, where you'll find extensions for Azure DevTest Labs, publishing to Google Play, Bower, Docker, HockeyApp, CodePush, FTP, ReactNative, Apache Cordova and much more. Again, spend a little time familiarizing yourself with what's possible. It's especially interesting to see tasks for the Google Play marketplace, Docker, HockeyApp and CodePush, among others. Along with the built-in deployment tasks, they demonstrate that you can easily place a task at the end of your build definition to deploy the results to other environments, including testing, staging and even production.

The MyDriving project provides some additional real-world build definition examples. For Xamarin.Android, it uses all of

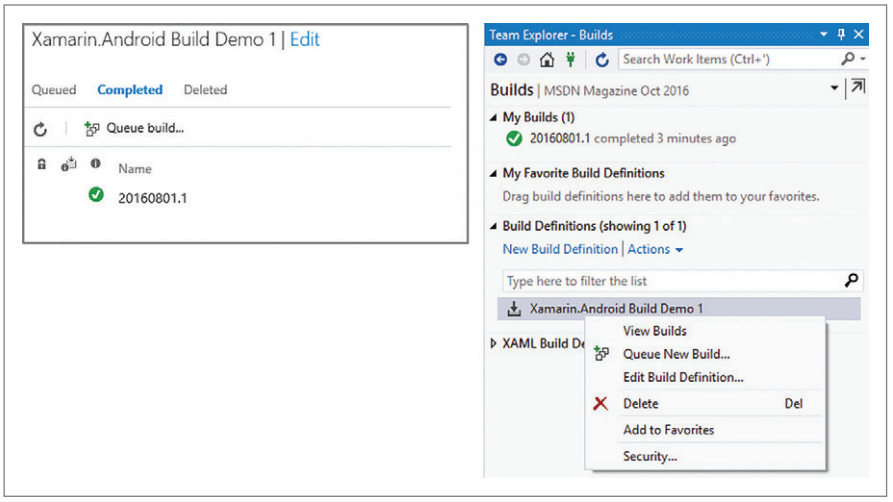


Figure 6 **TFBuild Results in Team Services (left) and Visual Studio (right)**

the same steps as discussed in the earlier example and adds a few more to replace tokens, update version names and numbers, use cURL to download a keystore, and deploy to Xamarin Test Cloud. The Universal Windows Platform and iOS build definitions are much the same, just with slightly different steps to manage details like signing certificates. And for the back-end ASP.NET code, MyDriving has a simple definition to run NuGet restore, invoke MSBuild, and publish the artifacts to a server. You can find a step-by-step breakdown of all these build definitions in Chapter 10 of the "MyDriving Reference Guide" at aka.ms/mydrivingdocs.

Deploying an Agent

A distinctive part of any definition is its list of "demands" that you can find on the General tab. The idea of demands is that if you try to queue a build to an agent that doesn't meet those demands, Team Services can give you an immediate error rather than attempting a hopeless build and making you pick through errors in build output. Thus, a Xamarin.Android definition will list demands like the Android SDK and Xamarin.Android. A build definition for iOS, in contrast, will list demands such as Xcode or Xamarin.iOS.

As noted before, hosted agents in Team Services are Windows machines, but only a Mac can meet the demands for iOS. How, then, do you get a build agent running on such a machine? One way is to use a service like MacinCloud, which is what's used in MyDriving. There's a great post on the Visual Studio ALM blog (bit.ly/2ajHtlz) that provides all the details about setting this up.

The other option is to install an agent on your own Mac. Detailed instructions are found on "Deploy an Agent on OS X" (bit.ly/2azm136), but let me share some of that experience here using the humble 5-year-old MacBook that's sitting on my desk. On that machine I downloaded the OS X agent and started the configuration. This prompted me to enter the URL of my Team Services account, followed by a personal access token obtained from my profile in the Team Services portal. The token is how the agent on the Mac identifies itself, given that a Mac doesn't otherwise know about my Microsoft account.

Once connected to Team Services, I was prompted for the pool to which to add the agent. As an example, I went to the Agent pool section in Team Services and created "On-Prem Mac," as shown in **Figure 7**. A few moments after I gave this name to the agent on the Mac, the agent appeared on Team Services, as shown in the inset.

With the agent connected, I made sure to run it on the Mac, which I could do either interactively or as a service. The "Deploy an Agent on OS X" page mentioned earlier describes all this. I then created a Xamarin.iOS build definition in Team Services and selected my Mac-based agent queue for the build (I also chose this in the build definition's General | Default agent queue setting). Finally, I queued a build through Team Services and—voilà—I see it running on my MacBook and see the build output directly in Team Services, just as with the hosted agent.

A Few Tips and Notes

- Remember that most projects will need a package restore step early in the build definition, such as the NuGet Installer step for Xamarin and .NET projects and the npm step for Apache Cordova projects. Build errors will remind you of this fact!
- By default, each step you add to a build definition will have its Control Options | Continue on error and Always run boxes unchecked. Checking the first means that the step isn't actually essential for subsequent steps, and shouldn't fail the build. The second means that the step should always run regardless of what happens with other steps. For example, you might have a step at the end of the definition to copy whatever artifacts were built, even if some didn't get built, or you might have a specialized cleanup task that should always run.
- When you do manage your own build agents, it's your responsibility to keep the software up-to-date. Build errors will typically tell you if there's a version mismatch somewhere.
- Remember to check the Options | Create Work Item on Failure box in the build definition to assign a bug to whoever committed code that triggered a CI build that failed. You can also find additional notification- and e-mail-related tasks in the Team Services marketplace with which to customize your process if you don't want to rely solely on work items.
- Have an idea for a build extension of your own? Check out the documentation at bit.ly/2avgl97.

Looking Ahead

As stated earlier in this article, Build is what transforms source code into the artifacts needed by the rest of the release pipeline (even if only for testing purposes). The next article in this series will explore the Release Management features of Visual Studio Team Services, which is how you can define and potentially automate any additional steps that must take place between a build and getting your app and services out to customers. Fortunately, much of what you've learned about Build definitions applies also to Release definitions, as the latter are also configured with discrete steps. All of this takes you ever closer to a fully automated release pipeline. ■

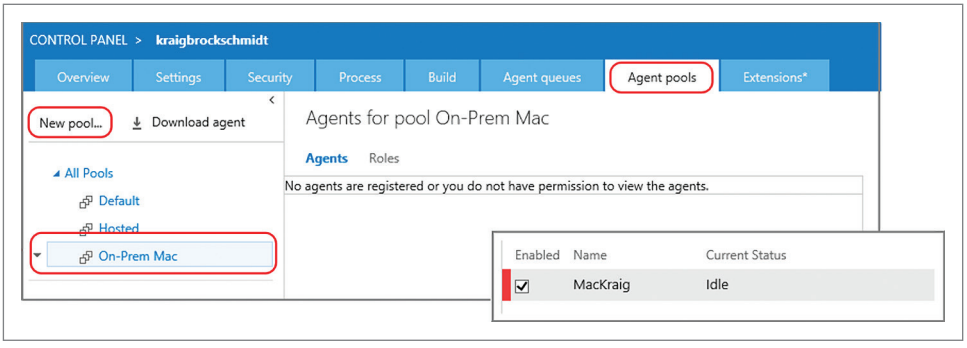


Figure 7 **Creating a New Agent Pool and How a New Agent Appears After Connecting (Inset)**

Kraig Brockschmidt *works as a senior content developer for Microsoft and is focused on DevOps for mobile apps. He's the author of "Programming Windows Store Apps with HTML, CSS and JavaScript" (two editions) from Microsoft Press and blogs on kraigbrockschmidt.com.*

Thanks to the following technical expert for reviewing this article: *Andy Lewis*

Orlando 2016

ROYAL PACIFIC RESORT AT UNIVERSAL ORLANDO

DEC 5-9

Visual Studio® LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

Journey into Code

Join us as we journey into real-world, practical education and training on the Microsoft Platform. Visual Studio Live! (VSLive!™) returns to warm, sunny Orlando for the conference more developers rely on to expand their .NET skills and the ability to build better applications.

twitter.com/live360
@live360

facebook.com
Search "Live 360"

linkedin.com
Join the "Live! 360" group!

EVENT PARTNERS

Microsoft Magenic

PLATINUM SPONSORS

HYPERGRID SMARTBEAR

GOLD SPONSORS

GOVERLAN IDERA

SUPPORTED BY

Visual Studio msdn magazine Redmond Channel Partner

LIVE! 360

TECH EVENTS WITH PERSPECTIVE

6 Great Conferences 1 Great Price

Visual Studio Live! Orlando is part of Live! 360, the Ultimate Education Destination. This means you'll have access to five (5) other co-located events at no additional cost:

SQL Server LIVE!
TRAINING FOR DBAs AND IT PROS

TECHMENTOR
IN-DEPTH TRAINING FOR IT PROS

Office & SharePoint LIVE!
ON-PREMISE, CLOUD & CROSS-PLATFORM TRAINING

Modern Apps LIVE!
MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT

NEW! APPDEV TRENDS
ENTERPRISE FOCUSED. CODE DRIVEN.

Six (6) events and hundreds of sessions to choose from – mix and match sessions to create your own, custom event line-up – it's like no other conference available today!

TURN THE PAGE FOR MORE EVENT DETAILS

Whether you are an

- Engineer
- Developer
- Programmer
- Software Architect
- Software Designer

You will walk away from this event having expanded your .NET skills and the ability to build better applications.

REGISTER TODAY AND **SAVE $400!**

Use promo code ORLSEP4 by October 5

Scan the QR code to register or for more event details.

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

ROYAL PACIFIC RESORT AT UNIVERSAL ORLANDO

DEC 5-9

Check Out the Additional Sessions for Devs, IT Pros, & DBAs at Live! 360

LIVE! 360

Office & SharePoint LIVE!

ON-PREMISE, CLOUD & CROSS-PLATFORM TRAINING

Office & SharePoint Live! features 12+ developer sessions, including:

- Workshop: A Beginner's Guide to Client Side Development in SharePoint - *Mark Rackley*
- Become a Developer Hero by Building Office Add-ins - *Bill Ayres*
- Utilizing jQuery in SharePoint - Get More Done Faster - *Mark Rackley*
- Using the Office UI Fabric - *Paul Schaeflein*
- Enterprise JavaScript Development Patterns - *Rob Windsor*
- Leveraging Angular2 to Build Office Add-ins - *Andrew Connell*
- Webhooks in Office 365 - *Paul Schaeflein*

SQL Server LIVE!

TRAINING FOR DBAs AND IT PROS

SQL Server Live! features 15+ developer sessions, including:

- What's New in SQL Server 2016 - *Leonard Lobel*
- Powerful T-SQL Improvements that Reduce Query Complexity - *Hugo Kornelius*
- Implementing Data Protection and Security in SQL Server 2016 - *Steve Jones*
- Welcome To The 2016 Query Store! - *Janis Griffin*
- Workshop: Big Data, Analytics and NoSQL: Everything You Wanted to Learn But Were Afraid to Ask - *Andrew Brust*

TECHMENTOR

IN-DEPTH TRAINING FOR IT PROS

TechMentor features IT Pro and DBA sessions, including:

- Workshop: 67 VMware vSphere Tricks That'll Pay for This Conference! - *Greg Shields*
- Secure Access Everywhere! Implementing DirectAccess in Windows Server 2016 - *Richard Hicks*
- Getting Started with Nano Server - *Jeffery Hicks*
- Creating Class-Based PowerShell Tools - *Jeffery Hicks*
- Harvesting the Web: Using PowerShell to Scrape Screens, Exploit Web Services, and Save Time - *Mark Minasi*
- PowerShell Unplugged: Stump Don - *Don Jones*
- Facing Increasing Malware Threats and a Growing Trend of BYO with a New Approach of PC Security - *Yung Chou*

ALM / DevOps	Cloud Computing	Mobile Client	Software Practices	Visual Studio / .NET Framework

START TIME	END TIME		
5:00 PM	8:00 PM	Pre-Conference Registration - Royal Pacific Resort Conference Center	
6:00 PM	9:00 PM	Dine-A-Round Dinner @ Universal CityWalk	
START TIME	**END TIME**		
8:00 AM	5:00 PM	**VSM01** Workshop: Distributed Cross-Platform Application Architecture - *Rockford Lhotka & Jason Bock*	
12:00 PM	1:00 PM	Lunch	
1:00 PM	5:00 PM	**VSM1** Workshop Continues	
5:00 PM	6:00 PM	EXPO Preview	
6:00 PM	7:00 PM	Live! 360 Keynote: To Be Announced	
START TIME	**END TIME**		
8:00 AM	9:00 AM	Visual Studio Live! / Modern Apps Live! Keynote: Topic To Be Announced	
9:00 AM	9:30 AM	Networking Break • Visit the EXPO	
9:30 AM	10:45 AM	**VST01** Building Applications with ASP.NET Core - *Scott Allen*	**VST02** Busy .NET Developer's Guide to Swift - *Ted Neward*
11:00 AM	12:15 PM	**VST05** Richer MVC Sites with Knockout JS - *Miguel Castro*	**VST06** Busy .NET Developer's Guide to Native iOS - *Ted Neward*
12:15 PM	2:00 PM	Lunch • Visit the EXPO	
2:00 PM	3:15 PM	**VST09** WCF & Web API: Can We All Just Get Along?!? - *Miguel Castro*	**VST10** Creating Great Looking Android Applications Using Material Design - *Kevin Ford*
3:15 PM	4:15 PM	Networking Break • Visit the EXPO	
4:15 PM	5:30 PM	**VST13** Busy Developer's Guide to Chrome Development - *Ted Neward*	**VST14** Using Visual Studio Tools for Apache Cordova to Create MultiPlatform Applications - *Kevin Ford*
5:30 PM	7:30 PM	Exhibitor Reception	
START TIME	**END TIME**		
8:00 AM	9:15 AM	**VSW01** Moving from Angular 1 to Angular 2 - *Ben Dewey*	**VSW02** The Future of Mobile Application Search - *James Montemagno*
9:30 AM	10:45 AM	**VSW05** Getting Started with Aurelia - *Brian Noyes*	**VSW06** Building Connected and Disconnected Mobile Applications - *James Montemagno*
10:45 AM	11:15 AM	Networking Break • Visit the EXPO	
11:15 AM	12:15 PM	Live! 360 Keynote: To Be Announced	
12:15 PM	1:45 PM	Birds-of-a-Feather Lunch • Visit the EXPO	
1:45 PM	3:00 PM	**VSW09** Living in a Command Line Web Development World (NPM, Bower, Gulp, and More) - *Ben Dewey*	**VSW10** Understanding the Windows Desktop App Development Landscape - *Brian Noyes*
3:00 PM	4:00 PM	Networking Break • Visit the EXPO • Expo Raffle @ 3:30 p.m.	
4:00 PM	5:15 PM	**VSW13** Securing Client JavaScript Apps - *Brian Noyes*	**VSW14** Let's Write a Windows 10 App: A Basic Introduction to Universal Apps - *Billy Hollis*
8:00 PM	10:00 PM	Live! 360 Dessert Luau - *Wantilan Pavilion*	
START TIME	**END TIME**		
8:00 AM	9:15 AM	**VSH01** Build Real-Time Websites and Apps with SignalR - *Rachel Appel*	**VSH02** Cognitive Services: Building Smart Applications with Computer Vision - *Nick Landry*
9:30 AM	10:45 AM	**VSH05** HTTP/2: What You Need to Know - *Robert Boedigheimer*	**VSH06** Building Business Apps on the Universal Windows Platform - *Billy Hollis*
11:00 AM	12:15 PM	**VSH09** TypeScript and ES2015 JumpStart - *John Papa*	**VSH10** A Developers Introduction to HoloLens - *Billy Hollis & Brian Randell*
12:15 PM	1:30 PM	Lunch on the Lanai	
1:30 PM	2:45 PM	**VSH13** All Your Tests Are Belong To Us - *Rachel Appel*	**VSH14** Developing Awesome 3D Apps with Unity and C# - *Adam Tuliper*
3:00 PM	4:15 PM	**VSH17** SASS and CSS for Developers - *Robert Boedigheimer*	**VSH18** From Oculus to HoloLens: Building Virtual & Mixed Reality Apps & Games - *Nick Landry*
4:30 PM	5:30 PM	Live! 360 Conference Wrap-Up - *Pacifica 6 - Andrew Brust (Moderator),*	
START TIME	**END TIME**		
8:00 AM	5:00 PM	**VSF01** Workshop: Angular 2 Bootcamp - *John Papa*	
12:00 PM	1:00 PM	Lunch	
1:00 PM	5:00 PM	**VSF01** Session Continues	

Speakers and sessions subject to change

Look for the FULL Live! 360 Pull-Out Agenda on Page 33

AGENDAS AT-A-GLANCE

Visual Studio LIVE! — EXPERT SOLUTIONS FOR .NET DEVELOPERS

Modern Apps LIVE! — MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT

Presented in Partnership with Magenic

APPDEV TRENDS — ENTERPRISE FOCUSED. CODE DRIVEN.

Web Client	Web Server	Windows Client	Modern Apps Live!	Agile	Containerization	Continuous Integration	Java	Mobile	Cloud

Pre-Conference: Sunday, December 4, 2016

Pre-Conference Workshops: Monday, December 5, 2016

VSM02 Workshop: Service Oriented Technologies - Designing, Developing, & Implementing WCF and the Web API - *Miguel Castro*	**VSM03** Workshop: DevOps in a Day - *Brian Randell*	**MAM01** Workshop: Building Modern Mobile Apps - *Brent Edwards & Kevin Ford*	**ADM01** Workshop: Building Teams - *Steve Green*	**ADM02** Workshop: One Codebase to Rule Them All: Xamarin - *Fabian Williams*
VSM2 Workshop Continues	**VSM3** Workshop Continues	**MAM01** Workshop Continues	**ADM01** Workshop Continues	**ADM02** Workshop Continues

Day 1: Tuesday, December 6, 2016

- *Tim Sneath, Principal Lead Program Manager, Visual Studio Platform, Microsoft*			**App Dev Trends Keynote: You Are the Future of Enterprise Java!** - *Reza Rahman, Speaker, Author, Consultant*	
VST03 What's New in Azure v2 - *Eric D. Boyd*	**VST04** Real World Scrum with Team Foundation Server 2015 & Visual Studio Team Services - *Benjamin Day*	**MAT01** Modern App Development: Transform How You Build Web and Mobile Software - *Rockford Lhotka*	**ADT01** Hacking Technical Debt - *Steve Green*	**ADT02** Java 8 Lambdas and the Streaming API - *Michael Remijan*
VST07 Overview of Power Apps - *Nick Pinheiro*	**VST08** Get Good at DevOps: Feature Flag Deployments with ASP.NET, WebAPI, & JavaScript - *Benjamin Day*	**MAT02** Architecture: The Key to Modern App Success - *Brent Edwards*	**ADT03** Are You A SOLID Coder? - *Steve Green*	**ADT04** PrimeFaces 5: Modern UI Widgets for Java EE - *Kito Mann*
VST11 Introduction to Next Generation of Azure PaaS – Service Fabric and Containers - *Vishwas Lele*	**VST12** To Be Announced	**MAT03** Manage Distributed Teams with Visual Studio Team Services and Git - *Brian Randell*	**ADT05** Agile Architecture - *Steve Green*	**ADT06** Full Stack Java with JSweet, Angular 2, PrimeNG, and JAX-RS - *Kito Mann*
VST15 Cloud Oriented Programming - *Vishwas Lele*	**VST16** Bringing DevOps to the Database - *Steve Jones*	**MAT04** Focus on the User Experience #FTW - *Anthony Handley*	**ADT07** Crafting Innovation - *Steve Green*	**ADT08** Who's Taking Out the Garbage? How Garbage Collection Works in the VM - *Kito Mann*

Day 2: Wednesday, December 7, 2016

VSW03 Managing Enterprise and Consumer Identity with Azure Active Directory - *Nick Pinheiro*	**VSW04** Improving Performance in .NET Applications - *Jason Bock*	**MAW01** DevOps, Continuous Integration, the Cloud, and Docker - *Dan Nordquist*	**ADW01** Stop Killing Requirements! - *Melissa Green*	**ADW02** Migrating Customers to Microsoft Azure: Lessons Learned From the Field - *Ido Flatow*
VSW07 Practical Internet of Things for the Microsoft Developer - *Eric D. Boyd*	**VSW08** I'll Get Back to You: Understanding Task, Await, and Asynchronous Methods - *Jeremy Clark*	**MAW02** Mobile Panel - *Kevin Ford, Rockford Lhotka, James Montemagno, & Jordan Matthiesen*	**ADW03** Meeting-Free Software Development in Distributed Teams - *Yegor Bugayenk*	**ADW04** The Essentials of Building Cloud-Based Web Apps with Azure - *Ido Flatow*
VSW11 To Be Announced	**VSW12** Learn to Love Lambdas (and LINQ, Too) - *Jeremy Clark*	**MAW03** C# Everywhere: How CSLA .NET Enables Amazing Cross-Platform Code Reuse - *Rockford Lhotka*	**ADW05** Introduction to Microsoft Office Graph - *Fabian Williams*	**ADW06** Building IoT and Big Data Solutions on Azure - *Ido Flatow*
VSW15 ARM Yourself for Azure Success - *Esteban Garcia*	**VSW16** Continuous Delivery on Azure: A/B Testing, Canary Releases, and Dark Launching - *Marcel de Vries*	**MAW04** Coding for Quality and Maintainability - *Jason Bock*	**ADW07** As You Think About Azure Databases, Think About DocumentDb - *Fabian Williams*	**ADW08** Where Does JavaScript Belong in the App Store? - *Jordan Matthiesen*

Day 3: Thursday, December 8, 2016

VSH03 C# Best Practices - *Scott Allen*	**VSH04** Application Insights: Measure Your Way to Success - *Esteban Garcia*	**MAH01** Modern Mobile Development: Build a Single App For iOS & Android with Xamarin Forms - *Kevin Ford*	**ADH01** From VMs to Containers: Introducing Docker Containers for Linux and Windows Server - *Ido Flatow*	**ADH02** Continuous Testing in a DevOps World - *Wayne Ariola*
VSH07 Debugging Your Way Through .NET with Visual Studio 2015 - *Ido Flatow*	**VSH08** The Ultimate Intro to Docker for Developers - *Adam Tuliper*	**MAH02** Universal Windows Development: UWP for PC, Tablet & Phone - *Brent Edwards*	**ADH03** CQRS 2.0 - Commands, Actors, and Events...Oh My! - *David Hoerster*	**DH04** Microservices as Chat Bots Are the Future - *Yegor Bugayenk*
VSH11 Exploring Microservices in a Microsoft Landscape - *Marcel de Vries*	**VSH12** Automated UI Testing for iOS and Android Mobile Apps - *James Montemagno*	**MAH03** Modern Web Development: ASP.NET MVC and Web API - *Allen Conway*	**ADH05** The Curious Case for the Immutable Object - *David Hoerster*	**ADH06** Continuous Integration May Have Negative Effects - *Yegor Bugayenk*
VSH15 Unit Testing Makes Me Faster: Convincing Your Boss, Your Co-Workers, and Yourself - *Jeremy Clark*	**VSH16** Writing Maintainable, X-Browser Automated Tests - *Marcel de Vries*	**MAH04** Modern Web Development: Building a Smart Web Client with TypeScript and Angular2 - *Allen Conway*	**ADH07** To Be Announced	**ADH08** Mobile DevOps Demystified with Xamarin, VSTS and HockeyApp - *Roy Cornelissen*
VSH19 User Experience Case Studies - Good and Bad - *Billy Hollis*	**VSH20** Debugging the Web with Fiddler - *Ido Flatow*	**MAH05** Using All That Data: Power BI to the Rescue - *Scott Diehl*	**ADH09** Get Started with Microsoft PowerApps - *Fabian Williams*	**ADH10** Overcoming the Challenges of Mobile Development in the Enterprise - *Roy Cornelissen*

Andrew Connell, Don Jones, Rockford Lhotka, Matthew McDermott, Brian Randell, & John K. Waters

Post-Conference Workshops: Friday, December 9, 2016

VSF02 Workshop: Building Modern Web Apps with Azure - *Eric D. Boyd & Brian Randell*	**MAF01** Workshop: Modern App Deep Dive: Xamarin, Responsive Web, UWP, CSLA .NET - *Jason Bock, Allen Conway, Brent Edwards & Kevin Ford*	**ADF01** Workshop: To Be Announced
VSF02 Session Continues	**MAF01** Session Continues	**ADF01** Session Continues

Create a Customizable FileSystemWatcher Windows Service

Diego Ordonez

The FileSystemWatcher class is a very powerful tool that's been a part of the Microsoft .NET Framework since version 1.1, and according to its official definition (bit.ly/2b8i0vQ), it "listens to the file system change notifications and raises events when a directory, or file in a directory, changes."

This class is able to detect events in the file system, such as create, modify, or delete files and folders; it's fully customizable; and its constructor accepts parameters like folder location and file extension to listen for, and a Boolean parameter to specify whether the listening process should work recursively through the folder structure. However, including those parameters in your source code isn't a good approach because they won't help when the application needs to include new folders and file extensions, which, moreover, will require coding, building and redeployment. Unless you're sure your application will hardly ever change those settings, a better idea is to implement a mechanism that can change the configuration without modifying the source code.

In this article I explore how to write an application that uses the FileSystemWatcher class just once, but then, via XML serialization, allows further modifications to the application's settings, such as folder names, file extensions and actions to be executed upon raising an event. In this way, all the changes can be easily achieved simply by updating an XML file and restarting the Windows service.

For simplicity's sake, I'm not going to explain the details about how to run this C# console application as a Windows service, but many resources are available online regarding this matter.

This article discusses:

- The structure of customized folder settings
- Starting the FileSystemWatcher process to listen for changes
- Starting and stopping FileSystemWatcher instances within a Windows service

Technologies discussed:

FileSystemWatcher Class, XML, Windows Services

Code download available at:

msdn.com/magazine/1016magcode

The Structure of Customized Folder Settings

Because I plan to deserialize the XML settings file into a well-structured C# class, the first component of the application must be the definition of the parameters FileSystemWatcher requires to operate. **Figure 1** shows the code that defines that class.

Now let's look at how an XML file can be translated into this C# class using the deserialization process. Please note that there won't be one single instance of the class CustomFolderSettings; instead there will be a list (List<CustomFolderSettings>) allowing the Windows service to listen for many different folder locations and file extensions.

Figure 2 shows an example of an XML settings file from which I can provide the FileSystemWatcher with all the arguments it needs to

work. It's important to understand at this point that the information contained in the XML file (**Figure 2**) will feed the C# class (**Figure 1**).

Let's take a closer look at the parameters contained in the XML file now. First, note that the XML root element is <ArrayOfCustomFolderSettings>, which allows as many elements <CustomFolderSettings> as required. This is the key to being able to concurrently monitor several folder locations and file extensions.

Second, notice that the parameter <FolderEnabled> is true for the first folder, but false for the second one. This is an easy way to disable one of the FileSystemWatchers without having to delete it from the XML file, meaning that even if the configuration is present, the class will omit it when it's running.

Finally, it's important to understand how to specify which action will be triggered upon detection of a file that has been created, deleted or modified, which is the final goal of the FileSystemWatcher class.

The parameter <ExecutableFile> contains the application that will be launched, in this example the DOS command line (CMD.EXE).

> Of course, the method needs to know where the XML settings file is, and I use the App.config file to define the location of the XML file.

The parameter <ExecutableArguments> contains the options that will be passed to the executable as arguments. Here's the example from **Figure 2**:

```
>/C echo It works properly for .ZIP extension -- File {0} &gt;
  c:\temp\it_ZIP_works.txt
```

This will translate into the following at running time:

```
CMD.EXE /C echo it works properly for .ZIP extension -- File
  d:\tests\file_modified_detected.doc > c:\temp\it_works_ZIP.txt
```

It will write the string into the file c:\temp\it_works_ZIP.txt, and the value {0} in the XML will be replaced by the actual name of the file FileSystemWatcher has detected. If you're familiar with the C# method string.Format, you won't have any problems figuring it out.

Well, at this point I have one XML configuration file and one C# class with matching attributes, so the next step is to deserialize the XML information into a list of classes (List<CustomFolderSettings>). **Figure 3** shows the method that performs this key step.

Once this method executes, a list containing all the required FileSystemWatcher instances will be available, so the next step is to start the FileSystemWatcher class, which starts the listening process.

Of course, the method needs to know where the XML settings file is, and I use the App.config file to define the location of the XML file. Here's the content of App.config:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="XMLFileFolderSettings" value=
      "C:\Work\CSharp_FileSystemW\CustomSettings.xml" />
  </appSettings>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
  </startup>
</configuration>
```

Figure 1 **Definition of the CustomFolderSettings Class**

```
/// <summary>
/// This class defines an individual type of file and its associated
/// folder to be monitored by the File System Watcher
/// </summary>
public class CustomFolderSettings
{
  /// <summary>Unique identifier of the combination File type/folder.
  /// Arbitrary number (for instance 001, 002, and so on)</summary>
  [XmlAttribute]
  public string FolderID { get; set; }

  /// <summary>If TRUE: the file type and folder will be monitored</summary>
  [XmlElement]
  public bool FolderEnabled { get; set; }

  /// <summary>Description of the type of files and folder location -
  /// Just for documentation purpose</summary>
  [XmlElement]
  public string FolderDescription { get; set; }

  /// <summary>Filter to select the type of files to be monitored.
  /// (Examples: *.shp, *.*, Project00*.zip)</summary>
  [XmlElement]
  public string FolderFilter { get; set; }

  /// <summary>Full path to be monitored
  /// (i.e.: D:\files\projects\shapes\ )</summary>
  [XmlElement]
  public string FolderPath { get; set; }

  /// <summary>If TRUE: the folder and its subfolders will be monitored</summary>
  [XmlElement]
  public bool FolderIncludeSub { get; set; }

  /// <summary>Specifies the command or action to be executed
  /// after an event has raised</summary>
  [XmlElement]
  public string ExecutableFile { get; set; }

  /// <summary>List of arguments to be passed to the executable file</summary>
  [XmlElement]
  public string ExecutableArguments { get; set; }

  /// <summary>Default constructor of the class</summary>
  public CustomFolderSettings()
  {
  }
}
```

Figure 2 **Structure of the XML Settings File**

```
<?xml version="1.0" encoding="utf-8"?>
<ArrayOfCustomFolderSettings xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <CustomFolderSettings FolderID="ExampleKML_files">
    <FolderEnabled>true</FolderEnabled>
    <FolderDescription>Files in format KML corresponding to the example project
      </FolderDescription>
    <FolderFilter>*.KML</FolderFilter>
    <FolderPath>C:\Temp\testKML\</FolderPath>
    <FolderIncludeSub>false</FolderIncludeSub>
    <ExecutableFile>CMD.EXE</ExecutableFile>
    <!-- The block {0} will be automatically replaced with the
      corresponding file name -->
    <ExecutableArguments>/C echo It works properly for .KML extension-- File {0}
      &gt; c:\temp\it_works_KML.txt</ExecutableArguments>
  </CustomFolderSettings>
  <CustomFolderSettings FolderID="ExampleZIP_files">
    <FolderEnabled>false</FolderEnabled>
    <FolderDescription>Files in format ZIP corresponding to the example project
      </FolderDescription>
    <FolderFilter>*.ZIP</FolderFilter>
    <FolderPath>C:\Temp\testZIP\</FolderPath>
    <FolderIncludeSub>false</FolderIncludeSub>
    <ExecutableFile>CMD.EXE</ExecutableFile>
    <!-- The block {0} will be automatically replaced with the
      corresponding file name -->
    <ExecutableArguments>/C echo It works properly for .ZIP extension -- File {0}
      &gt; c:\temp\it_works_ZIP.txt</ExecutableArguments>
  </CustomFolderSettings>
</ArrayOfCustomFolderSettings>
```

Figure 3 **Deserialization of the XML Settings File**

```
/// <summary>Reads an XML file and populates a list of
<CustomFolderSettings> </summary>
private void PopulateListFileSystemWatchers()
{
  // Get the XML file name from the App.config file
  fileNameXML = ConfigurationManager.AppSettings["XMLFileFolderSettings"];

  // Create an instance of XMLSerializer
  XmlSerializer deserializer =
    new XmlSerializer(typeof(List<CustomFolderSettings>));
  TextReader reader = new StreamReader(fileNameXML);
  object obj = deserializer.Deserialize(reader);

  // Close the TextReader object
  reader.Close();

  // Obtain a list of CustomFolderSettings from XML Input data
  listFolders = obj as List<CustomFolderSettings>;
}
```

Figure 4 **Initialization of the FileSystemWatcher Instances**

```
/// <summary>Start the file system watcher for each of the file
/// specification and folders found on the List<>/// </summary>
private void StartFileSystemWatcher()
{
  // Creates a new instance of the list
  this.listFileSystemWatcher = new List<FileSystemWatcher>();

  // Loop the list to process each of the folder specifications found
  foreach (CustomFolderSettings customFolder in listFolders)
  {
    DirectoryInfo dir = new DirectoryInfo(customFolder.FolderPath);

    // Checks whether the folder is enabled and
    // also the directory is a valid location
    if (customFolder.FolderEnabled && dir.Exists)
    {
      // Creates a new instance of FileSystemWatcher
      FileSystemWatcher fileSWatch = new FileSystemWatcher();

      // Sets the filter
      fileSWatch.Filter = customFolder.FolderFilter;

      // Sets the folder location
      fileSWatch.Path = customFolder.FolderPath;

      // Sets the action to be executed
      StringBuilder actionToExecute = new StringBuilder(
        customFolder.ExecutableFile);

      // List of arguments
      StringBuilder actionArguments = new StringBuilder(
        customFolder.ExecutableArguments);

      // Subscribe to notify filters
      fileSWatch.NotifyFilter = NotifyFilters.LastWrite | NotifyFilters.FileName |
        NotifyFilters.DirectoryName;

      // Associate the event that will be triggered when a new file
      // is added to the monitored folder, using a lambda expression
      fileSWatch.Created += (senderObj, fileSysArgs) =>
        fileSWatch_Created(senderObj, fileSysArgs,
         actionToExecute.ToString(), actionArguments.ToString());

      // Begin watching
      fileSWatch.EnableRaisingEvents = true;

      // Add the systemWatcher to the list
      listFileSystemWatcher.Add(fileSWatch);

      // Record a log entry into Windows Event Log
      CustomLogEvent(String.Format(
        "Starting to monitor files with extension ({0}) in the folder ({1})",
        fileSWatch.Filter, fileSWatch.Path));
    }
  }
}
```

It's important to remember that any changes in the XML settings file or in the App.config file will require restarting the Windows service in order for those changes to be applied.

Starting the FileSystemWatcher Process (Listening for Changes)

At this point, all the settings required for the several (or at least one) instances of the FileSystemWatcher are available in the list created in **Figure 3**.

Now it's time to start the listening process. For this, I need to loop through the list and start the instances one-by-one. The code in **Figure 4** shows how to perform the initialization process and how to assign all the parameters retrieved from the XML file.

This application is designed to be run as a Windows service, so I need a way to start or stop FileSystemWatcher instances automatically when the Windows service starts, stops or restarts.

In this code, the FileSystemWatcher is listening only for a creation event; however, other events are available, as well, such as Deleted and Renamed.

I want to especially point to the line where a function subscribes to the FileSystemWatcher Created event. Here, I use a lambda expression for an important reason: Because I have a list of instances of the FileSystemWatcher class, I need to associate a specific executable to each instance. If I handle this differently (that is, by not using a lambda expression but directly assigning the function), only the last executable will be kept and all the FileSystemWatcher instances will perform the same action.

Figure 5 shows the code for the function that actually performs the action based on individual criteria for each single instance of the FileSystemWatcher.

And, finally, **Figure 6** shows the ExecuteCommandLineProcess function, which is a very standard way to execute command-line instructions (a DOS console).

Starting and Stopping FileSystemWatcher Within a Windows Service

As initially stated, this application is designed to be run as a Windows service, so I need a way to start or stop FileSystemWatcher instances automatically when the Windows service starts, stops or restarts. Even though I'm not going to dig into the Windows Service definition here, it's worth mentioning the two main methods of the Windows service implementation: OnStart and OnStop.

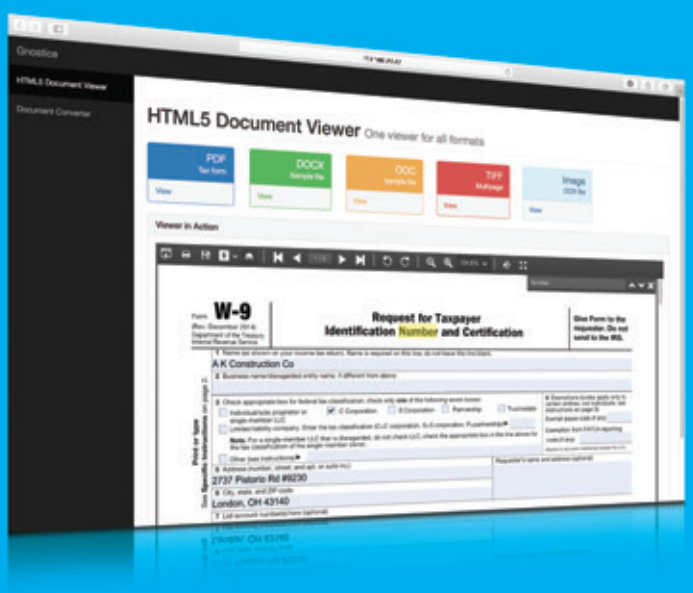
Document Technology for Everybody

PDF DOCX DOC RTF PPTX XLSX XLS TIFF MTIFF JPEG PNG BMP WMF EMF

HTML5 X Web MVC WPF Win API API Java VCL FMX

Create | View | Print | Edit | Convert | Digitize

Get the Power of Great Design



Interactively Fill PDF Forms In Your Web App

- Responsive HTML5 control, automatically adjusts for Mobile, Desktop and Pad/Tablet.
- 100% Independent. No browser plug-ins required. No ActiveX.
- Single viewer for PDF, Office documents, text files and Images.
- Text Search, On-the-fly OCR on images, and more…
- Full-fledged client-side JavaScript to configure and perform all operations.
- TypeScript Support.

Royalty-Free Licensing

Gnostice™
Smart needs...Smarter solutions...

Download free trial from:
www.gnostice.com

Figure 5 **Performing an Action Based on the Criteria for Each Instance**

```
/// <summary>This event is triggered when a file with the specified
/// extension is created on the monitored folder</summary>
/// <param name="sender">Object raising the event</param>
/// <param name="e">List of arguments - FileSystemEventArgs</param>
/// <param name="action_Exec">The action to be executed upon detecting a
change in the File system</param>
/// <param name="action_Args">arguments to be passed to the executable
(action)</param>
void fileSWatch_Created(object sender, FileSystemEventArgs e,
  string action_Exec, string action_Args)
{
  string fileName = e.FullPath;

  // Adds the file name to the arguments. The filename will be placed in lieu of {0}
  string newStr = string.Format(action_Args, fileName);

  // Executes the command from the DOS window
  ExecuteCommandLineProcess(action_Exec, newStr);
}
```

Figure 6 **Executing Command-Line Instructions**

```
/// <summary>Executes a set of instructions through the command window</summary>
/// <param name="executableFile">Name of the executable file or program</param>
/// <param name="argumentList">List of arguments</param>
private void ExecuteCommandLineProcess(string executableFile, string argumentList)
{
  // Use ProcessStartInfo class
  ProcessStartInfo startInfo = new ProcessStartInfo();
  startInfo.CreateNoWindow = true;
  startInfo.UseShellExecute = false;
  startInfo.FileName = executableFile;
  startInfo.WindowStyle = ProcessWindowStyle.Hidden;
  startInfo.Arguments = argumentList;

try
  {
    // Start the process with the info specified
    // Call WaitForExit and then the using-statement will close
    using (Process exeProcess = Process.Start(startInfo))
    {
      exeProcess.WaitForExit();

      // Register a log of the successful operation
      CustomLogEvent(string.Format(
        "Succesful operation --> Executable: {0} --> Arguments: {1}",
        executableFile, argumentList));
    }
  }
  catch (Exception exc)
  {
    // Register a Log of the Exception
  }
}
```

Figure 7 **Stopping the FileSystemWatcher**

```
/// <summary>Event automatically fired when the service is stopped by
Windows</summary>
protected override void OnStop()
{
  if (listFileSystemWatcher != null)
  {
    foreach (FileSystemWatcher fsw in listFileSystemWatcher)
    {
      // Stop listening
      fsw.EnableRaisingEvents = false;

      // Dispose the Object
      fsw.Dispose();
    }

    // Clean the list
    listFileSystemWatcher.Clear();
  }
}
```

Initially, every time the Windows service starts, it has to perform two actions: Populate the list of FileSystemWatcher instances from the XML file (**Figure 3**), and then start the instances (**Figure 4**).

Here's the code required to start the process from the Windows service:

```
/// <summary>Event automatically fired when the service is started by
Windows</summary>
/// <param name="args">array of arguments</param>
protected override void OnStart(string[] args)
{
  // Initialize the list of FileSystemWatchers based on the XML configuration file
  PopulateListFileSystemWatchers();

  // Start the file system watcher for each of the file specification
  // and folders found on the List<>
  StartFileSystemWatcher();
}
```

FileSystemWatcher is a powerful class that allows you to monitor (listen to) changes occurring in the file system, such as creating, deleting, and renaming files and folders, as well as modifying them.

And, finally, the method in **Figure 7** implements the logic to stop the FileSystemWatcher; it requires stopping or restarting the Windows service.

Wrapping Up

FileSystemWatcher is a powerful class that allows you to monitor (listen to) changes occurring in the file system, such as creating, deleting, and renaming files and folders, as well as modifying them. This application, which is intended to run as a Windows service, has been designed to allow for easy modification of the files and folders to be monitored, including file extensions. The approach I followed uses a very handy concept available in the .NET Framework, serialization and deserialization, making it possible to feed the FileSystemWatcher class from an XML file without requiring any change to the source code. Instead, after any modification in the XML settings file, it's just a matter of restarting the Windows service and, voilà, the changes are applied. ■

Diego Ordonez *is a civil engineer with more than 15 years of experience in IT working mainly with GIS and CAD technologies as an analyst, developer and architect. He is a Microsoft Certified Professional Developer in C#, ASP.NET, ADO.NET, SQL Server and he really enjoys learning and applying technologies around the .NET Framework. He lives in Calgary, Alberta, Canada, with his wife and two lovely daughters and works for Altus Geomatics as a GIS team lead (bit.ly/2aWfi34).*

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

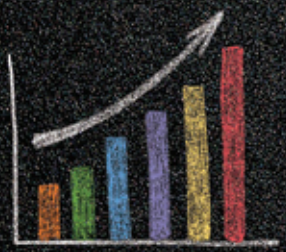
Spreadsheets Made Easy.

Fastest Calculations

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

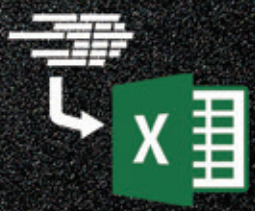
Powerful Controls

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.



Scalable Reporting

Easily create richly formatted Excel reports without Excel from any ASP.NET, Windows Forms, WPF or Silverlight application.

Download your free fully functional evaluation at SpreadsheetGear.com

SpreadsheetGear

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

JAMES MCCAFFREY

ANOVA with C#

Analysis of variance (ANOVA) is a classical statistics technique that's used to infer if the means (averages) of three or more groups are all equal, in situations where you only have sample data. For example, suppose there are three different introductory computer science classes at a university. Each class is taught by the same teacher, but uses a different textbook and a different teaching philosophy. You want to know if student performance is the same or not.

You have an exam that evaluates computer science proficiency on a one-to-15 scale, but because the exam is very expensive and time-consuming, you can give the exam to only six randomly selected students in each class. You administer the exam and perform ANOVA on the samples to infer if the means of all three classes are the same or not.

If you're new to ANOVA, the name of the technique may be mildly confusing because the goal is to analyze the means of data sets. ANOVA is named as it is because behind the scenes it analyzes variances to make inferences about means.

A good way to get an idea of what ANOVA is and to see where this article is headed is to take a look at the demo program in **Figure 1**. The demo sets up hardcoded scores for three groups. Notice that there are only four scores in Group1 and only five scores in Group3—it's quite common for sample sizes to be unequal because test subjects can drop out or data can be lost or corrupted.

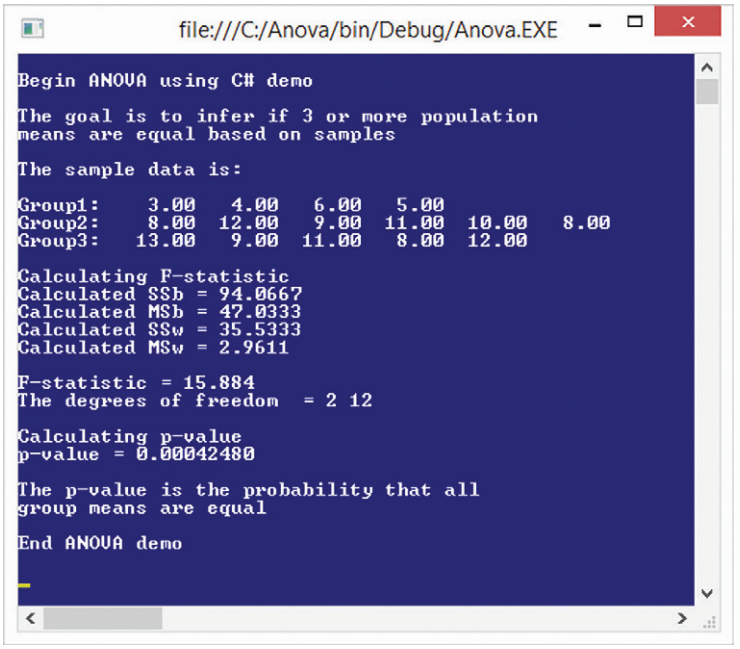


Figure 1 **ANOVA with C# Demo Program**

There are two main steps to ANOVA. In the first step, an F-statistic value and a pair of values called the "degrees of freedom" (df) are calculated using the sample data. In the second step, the values of F and df are used to determine the probability that all population means are the same (the p-value). The first step is relatively easy. The second step is very difficult.

In the demo, the value of F is 15.884. In general, the larger F is, the less likely it is that all population means are equal. I'll explain why df = (2, 12) shortly. Using F and df, the p-value is calculated to be 0.000425. This is very small, so you'd conclude that the population means are likely not all the same. At this point, you could perform additional statistical tests to determine which population means are different from each other. For the demo data, it appears that Group1 (sample mean = 4.50) is worse than Group2 (mean = 9.67) and Group3 (mean = 10.60).

Code download available at msdn.com/magazine/1016magcode.

The Demo Program

To create the demo program, I launched Visual Studio, clicked on File | New | Project and selected the C# Console Application option. The demo program has no significant .NET dependencies, so any version of Visual Studio will work. After the template code loaded in the Solution Explorer window, I right-clicked on file Program.cs and renamed it to AnovaProgram.cs and allowed Visual Studio to automatically rename class Program for me.

At the top of the editor window, I deleted all unnecessary using statements, leaving just the one that references the top-level System namespace. The overall structure of the program is shown in **Figure 2**. The demo program is too long to present in its entirety, but the complete demo source code is available in the download that accompanies this article.

Static method Fstat computes and returns an F-statistic based on data stored in an array-of-arrays object. The method also calculates and returns two df values in an array out-parameter. Function ShowData is just a little helper function to display the sample means.

The remaining five methods are all used to calculate the p-value. Method QF is the primary method. It calls method PF, which in turn calls method BetaInc, which in turn calls methods BetaIncCf and LogGamma.

LIVE! 360 | ORLANDO ROYAL PACIFIC RESORT AT UNIVERSAL ORLANDO | DEC 5-9

A NEW CONFERENCE FOR SOFTWARE DEVELOPERS

APPDEV TRENDS

ENTERPRISE FOCUSED. CODE DRIVEN.

Powering Enterprise Development

App Dev Trends, brought to you by ADTmag.com, is a new technology conference focused on the makers and maintainers of the purpose-designed software that Power organizations in virtually every industry in the world—in other words, enterprise software professionals! You Power your company. It's our job to Power you!

This event is for:

- down-in-the-trenches developers
- team leaders
- entire software development teams

Track topics include:

- Agile
- Cloud
- Mobility
- Java
- Containerization
- Continuous Integration

REGISTER BY OCTOBER 5 AND **SAVE $400!**

Use promo code ADTSEP1

Scan the QR code to register or for more event details.

A Part of Live! 360: The Ultimate Education Destination

6 Great Conferences, 1 Great Price

Visual Studio LIVE! | Modern Apps LIVE! | TECHMENTOR | Office & SharePoint LIVE! | SQL Server LIVE! | APPDEV TRENDS NEW!

APPDEVTRENDS.COM

APPDEV TRENDS ENTERPRISE FOCUSED. CODE DRIVEN. | ADTMAG APPLICATION DEVELOPMENT TRENDS

PRODUCED BY 1105 MEDIA

After some preliminary WriteLine messages, the Main method sets up and displays the sample data:

```
double[][] data = new double[3][]; // 3 groups
data[0] = new double[] { 3, 4, 6, 5 };
data[1] = new double[] { 8, 12, 9, 11, 10, 8 };
data[2] = new double[] { 13, 9, 11, 8, 12 };
string[] colNames = new string[] {  "Group1", "Group2", "Group3" };
ShowData(data, colNames);
```

In a non-demo scenario, your data would likely be stored in a text file and you'd write a helper function to read and load the data into an array-of-arrays.

The F-statistic and df are calculated like so:

```
int[] df = null;
double F = Fstat(data, out df);
```

> In short, when performing an ANOVA, the calling statements are very simple. But there's a lot of work that goes on behind the scenes.

For ANOVA, the df for a data set is a pair of values. The first value is K - 1 where K is the number of groups, and the second value is N - K where N is the total number of sample values. So for the demo data, df = (K-1, N-K) = (3-1, 15-3) = (2, 12).

The p-value is computed and displayed like this:

```
double pValue = QF(df[0], df[1], F);
Console.Write("p-value = ");
```

In short, when performing an ANOVA, the calling statements are very simple. But there's a lot of work that goes on behind the scenes.

Calculating the F-Statistic

Calculating the value of an F-statistic has several sub-steps. Suppose the sample data values are the ones from the demo:

```
Group1: 3.00, 4.00, 6.00, 5.00
Group2: 8.00, 12.00, 9.00, 11.00, 10.00, 8.00
Group3: 13.00, 9.00, 11.00, 8.00, 12.00
```

Figure 2 **Demo Program Structure**

```
using System;
namespace Anova
{
  class AnovaProgram
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Begin ANOVA using C# demo");
      // Set up sample data
      // Use data to calculate F and df
      // Use F and df to calculate p-value
      Console.WriteLine("End ANOVA demo");
    }

    static double Fstat(double[][] data, out int[] df) { . . }
    static double LogGamma(double z) { . . }
    static double BetaIncCf(double a, double b, double x) { . . }
    static double BetaInc(double a, double b, double x) { . . }
    static double PF(double a, double b, double x) { . . }
    static double QF(double a, double b, double x) { . . }
    static void ShowData(double[][] data, string[] colNames) { . . }
  }
}
```

The first sub-step is to calculate the means of each group, and the overall mean of all sample values. For the demo data:

```
means[0] = (3.0 + 4.0 + 6.0 + 5.0) / 4 = 4.50
means[1] = (8.0 + 12.0 + 9.0 + 11.0 + 10.0 + 8.0) / 6 = 9.67
means[2] = (13.0 + 9.0 + 11.0 + 8.0 + 12.0) / 5 = 10.60
gMean = (3.0 + 4.0 + . . . + 12.0) / 15 = 8.60
```

The definition of method Fstat starts with:

```
static double Fstat(double[][] data, out int[] df)
{
  int K = data.Length; // Number groups
  int[] n = new int[K]; // Number items each group
  int N = 0; // total number data points
  for (int i = 0; i < K; ++i) {
    n[i] = data[i].Length;
    N += data[i].Length;
  }
...
```

At this point, local array n has the number of values in each group, K has the number of groups, and N is the total number of values in all groups. Next, the group means are calculated into an array named means, and the overall grand mean is calculated into variable gMean:

```
double[] means = new double[K];
double gMean = 0.0;
for (int i = 0; i < K; ++i) {
  for (int j = 0; j < data[i].Length; ++j) {
    means[i] += data[i][j];
    gMean += data[i][j];
  }
  means[i] /= n[i];
}
gMean /= N;
```

The next sub-step is to calculate the "sum of squares between groups" (SSb) and "mean square between groups" (MSb). SSb is the weighted sum of squared differences between each group mean and the overall mean. MSb = SSb / (K-1) where K is the number of groups. For the demo data:

```
SSb = (4 * (4.50 - 8.60)^2) + (6 * (9.67 - 8.60)^2) + (5 * (10.60 - 8.60)^2) = 94.07
MSb = 94.07 / (3-1) = 47.03
```

The code that calculates SSb and MSb is:

```
double SSb = 0.0;
for (int i = 0; i < K; ++i)
  SSb += n[i] * (means[i] - gMean) * (means[i] - gMean);
double MSb = SSb / (K - 1);
```

The next sub-step is to calculate the "sum of squares within groups" (SSw) and the "mean square within groups (MSw). SSw is the sum of squared differences between each sample value and its group mean. MSw = SSw / (N-K). For the demo data:

```
SSw = (3.0 - 4.50)^2 + . . + (8.0 - 9.67)^2 + . . + (12.0 - 10.60)^2 = 35.53
MSw = 35.53 / (15-3) = 2.96
```

The code that calculates SSw and MSw is:

```
double SSw = 0.0;
for (int i = 0; i < K; ++i)
  for (int j = 0; j < data[i].Length; ++j)
    SSw += (data[i][j] - means[i]) * (data[i][j] - means[i]);
double MSw = SSw / (N - K);
```

The final sub-step is to calculate the two df values and the F-statistic. The two df values are K - 1, and N - K. And F = MSb / MSw. For the demo data:

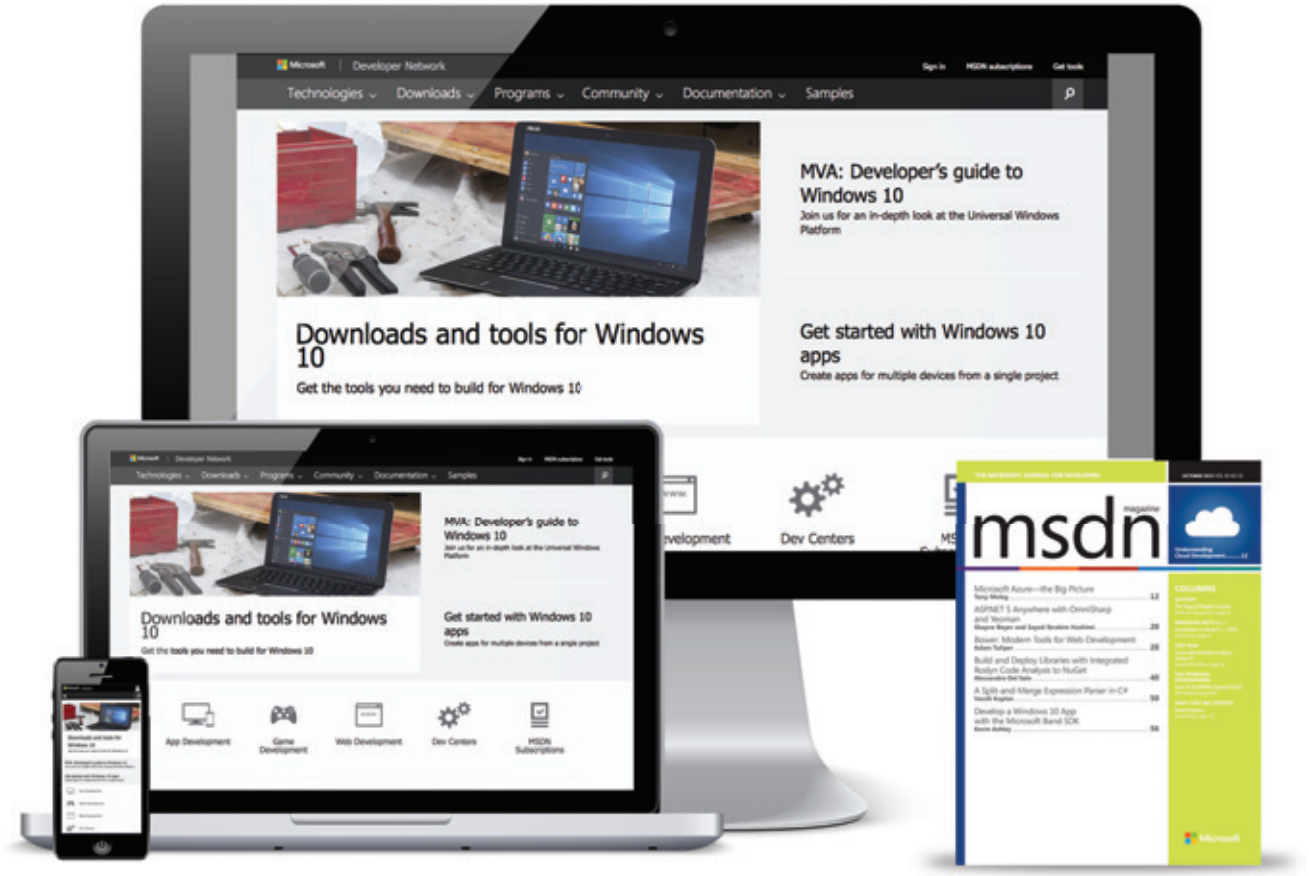
```
df = (K-1, N-K) = (3-1, 15-3) = (2, 12)
F = 47.03 / 2.96 = 15.88.
```

The demo code that calculates df and F is:

```
...
  df = new int[2];
  df[0] = K - 1;
  df[1] = N - K;
  double F = MSb / MSw;
  return F;
} // Fstat
```

msdn magazine

Where you need us most.



Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

LIVE! 360
TECH EVENTS WITH PERSPECTIVE

MSDN.microsoft.com

I think you'll agree that calculating an F-statistic and df values from a set of data is mechanical and relatively easy once you know the math equations.

Calculating the p-value

Converting an F-statistic and df values into a p-value that tells you the probability that all population means are equal based on the sample data that produce F and df is simple in principle, but extremely difficult in practice. I'll explain as briefly as possible, leaving out an enormous amount of detail that would require a huge amount of additional explanation. Take a look at the graph in **Figure 3**.

Each possible pair of df values determines a graph called the F-distribution. The shape of an F-distribution can vary wildly based on the values of df. The graph in **Figure 3** shows an F-distribution for df = (4, 12). I used df = (4, 12) rather than the df = (2, 12) from the demo data because the shape of the df = (2,12) F-distribution is very atypical.

> The total area under any F-distribution is exactly 1.0.

The total area under any F-distribution is exactly 1.0. If you know the value of the F-statistic, then the p-value is the area under the F-distribution from F to positive infinity. Somewhat confusingly, the area under the F-distribution from zero to the F-statistic is often called PF, and the area under the F-distribution from the F-statistic to positive infinity (representing the p-value) is often called QF. Because the total area under the distribution is 1, PF + QF = 1. It turns out that it's a bit easier to calculate PF than QF, so to find the p-value (QF), you typically calculate PF then subtract that from 1 to get QF.

Calculating PF is brutally difficult but, luckily, magic estimation equations have been known for decades. These math equations, and hundreds of others, can be found in a famous reference, "Handbook of Mathematical Functions" by M. Abramowitz and I.A. Stegun. The book is often called simply "A&S" among scientific programmers. Each A&S equation has an ID number.

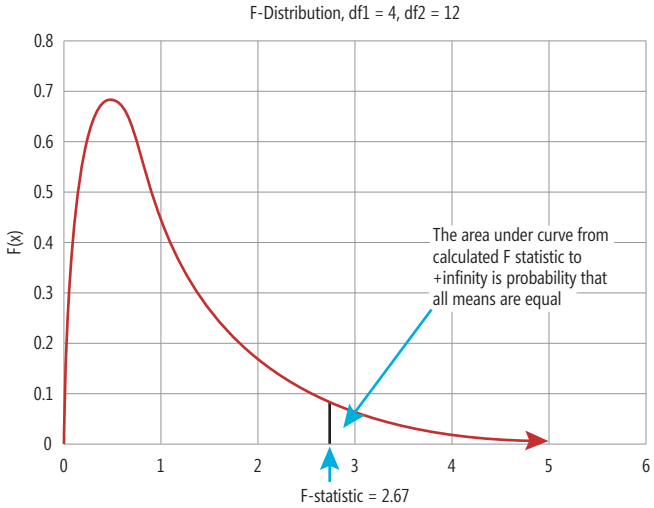


Figure 3 **Calculating the p-Value from an F-Statistic and df Values**

In the demo, method PF is really just a wrapper around method BetaInc:

```
static double PF(double a, double b, double x)
{
  double z = (a * x) / (a * x + b);
  return BetaInc(a / 2, b / 2, z);
}
```

The name of method BetaInc stands for "incomplete Beta." Method BetaInc uses A&S equations 6.6.2 and 26.5.8. Those equations call a LogGamma function and a BetaIncCf function. The LogGamma function is extremely difficult to explain and to implement. Briefly, the mathematical Gamma function extends the notion of factorial to real-valued numbers. Just like factorials, the values of the Gamma function can become astronomically large, so to handle them it's common to compute the log of Gamma to keep values smaller.

Calculating LogGamma is very tricky and there are several algorithms you can use. The demo program uses an algorithm called the Lanczos approximation with (g=5, n=7). The A&S reference has different algorithms that can calculate LogGamma, but the Lanczos approximation, which was not known when A&S was published, gives more accurate results.

The name of method BetaIncCf stands for "incomplete Beta computed by continued fraction." The demo program uses A&S equation 26.5.8 for method BetaIncCf.

Wrapping Up

An ANOVA test makes three mathematical assumptions: that the group data items are mathematically independent; that the population data sets are distributed normally (as in the Gaussian distribution); and that the population data sets have equal variances.

There are several ways you can test these assumptions, but interpreting their results is challenging. The problem is that it's highly unlikely that real-world data is exactly normal and has exactly equal variances, though ANOVA still works when data is somewhat non-normal or has non-equal variances. The bottom line is that it's extremely difficult to prove ANOVA assumptions so you should be very conservative when interpreting results.

ANOVA is closely related to the t-test. The t-test determines if the population means of exactly two groups are equal, in situations where you have only sample data. So, if you have three groups, as in the demo, instead of using ANOVA, you could conceivably perform three t-tests, comparing groups 1 and 2, groups 1 and 3, and groups 2 and 3. However, this approach isn't recommended because it introduces what's called a Type 1 error (a false positive).

The kind of ANOVA explained in this article is called one-way (or one-factor) ANOVA. A different technique, called two-way ANOVA, is used when there are two factors.

ANOVA is based on the calculated value of an F-statistic from a data set. There are other statistical tests that use an F-statistic. For example, you can use an F-statistic to infer if the variances of two groups of data are the same. ■

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 jammc@microsoft.com.*

Thanks to the following Microsoft technical experts who reviewed this article: *Chris Lee and Kirk Olynk*

How To Be MEAN: Exploring Yeoman

Welcome back, MEANers.

As I mentioned in a previous column, it's time to pull a comic-book move and engage in a little "retroactive continuity"—a common move whenever the story needs to change its past to better support its future. In this particular case, the ret-con you need to engage in is to make use of some of the tools that you should've used from the beginning, but didn't, because I deemed it necessary to walk through some of the parts in a step-by-step fashion before effectively hiding them behind the tools.

One thing I've heard through some of your e-mails, for example, is how much the MEAN development experience differs from that of the traditional .NET experience. One commenter went so far as to point out that with Visual Studio, you have all the project templates that can take much of the burden of organizing the source away from you. ASP.NET MVC, for example, decided a long time ago what directory controllers would live in, so that any ASP.NET MVC developer walking up to any ASP.NET MVC project will know exactly where everything lives.

> Like almost everything in the JavaScript universe, Yeoman is a command-line tool that's installed via npm.

On that note, I begin my discussion on Yeoman, the ECMAScript scaffolding tool, which serves the same purpose as the project template facility in Visual Studio. (For those who aren't aware of the etymology of the term "yeoman," one such dictionary definition defines it as "a servant in a royal or noble household, ranking between a sergeant and a groom or a squire and a page." For this reason, the Yeoman tool, documentation and site tend to pass themselves off as a Cockney English bloke.)

Get Your Yeoman on, Yo

Like almost everything in the JavaScript universe, Yeoman is a command-line tool that's installed via npm. The documentation for Yeoman is available at yeoman.io, but it's already obvious what the first step will be: "npm install --g yo." Once completed, this will put the "yo" command-line tool on the PATH, and an easy way to check that it's installed is to simply run it. Running "yo --help" brings up the usual list of options, but running "yo" by itself will actually be more interesting: It brings up an interactive command-line menu of options that Yeoman can execute on your behalf, as shown in **Figure 1**.

When freshly installed, Yeoman will not have any generators already installed (contrary to my list in **Figure 1** of a few already there), but scrolling up and down (using the arrow keys) will also show a few other options, such as "Install a generator," "Find some help" or "Get me out of here!" Yeoman is nothing if not obvious.

The term "generator" is an apt and appropriate term; like Visual Studio, Yeoman doesn't actually know how to generate anything itself. Instead, it relies on "generators," community-contributed packages that consist of scripts and templates that can interrogate the user for specifics of what to generate. In order for Yeoman to use one of these generators, however, it must be installed on the local system. In addition, in order for it to be installed, you need to figure out which of the almost 4,000 generators (as of this writing) available you want to use.

Finding Generators, Yo

While it's certainly possible to let Yeoman search npm for generators (one of the options presented by the command line when it's run), it's usually easier to let search engines manage that for you, or to simply browse for the generator on the Yeoman site (yeoman.io/generators). Therefore, for example, if you want to make use of Yeoman to scaffold out a new MEAN application, you need to find

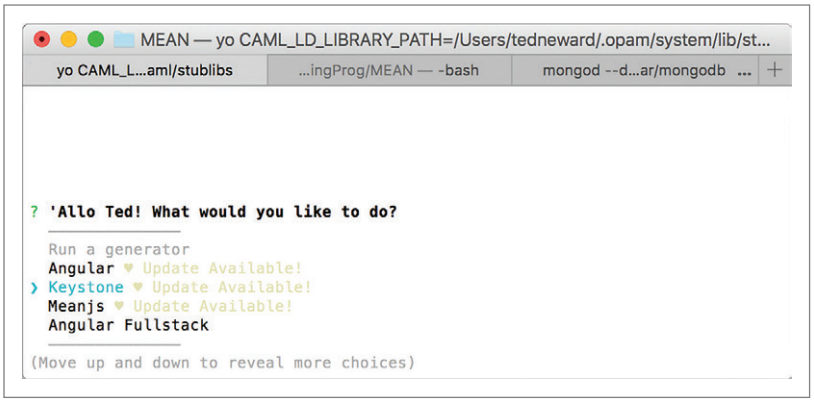


Figure 1 **Yeoman with Execution Options**

one that does Angular (v1 for now) and MongoDB. As it turns out, as of this writing, the one that's by far the most popular, "angular-fullstack," is right near the top of the list. However, if it wasn't, or if there was something different desired (such as a generator that did React, or for a Chrome extension, or even an ASP.NET Core generator), you could use the Yeoman generators page to search for it. For example, you might want to split the code into two projects, one for the back-end Web API, and one for the front-end Angular client. That would suggest that you want an Express-plus-MongoDB project (sometimes called a MEN project), which has a couple of generators available, including "express-api" or "node-express-mongo"), plus another generator that can build an Angular front end.

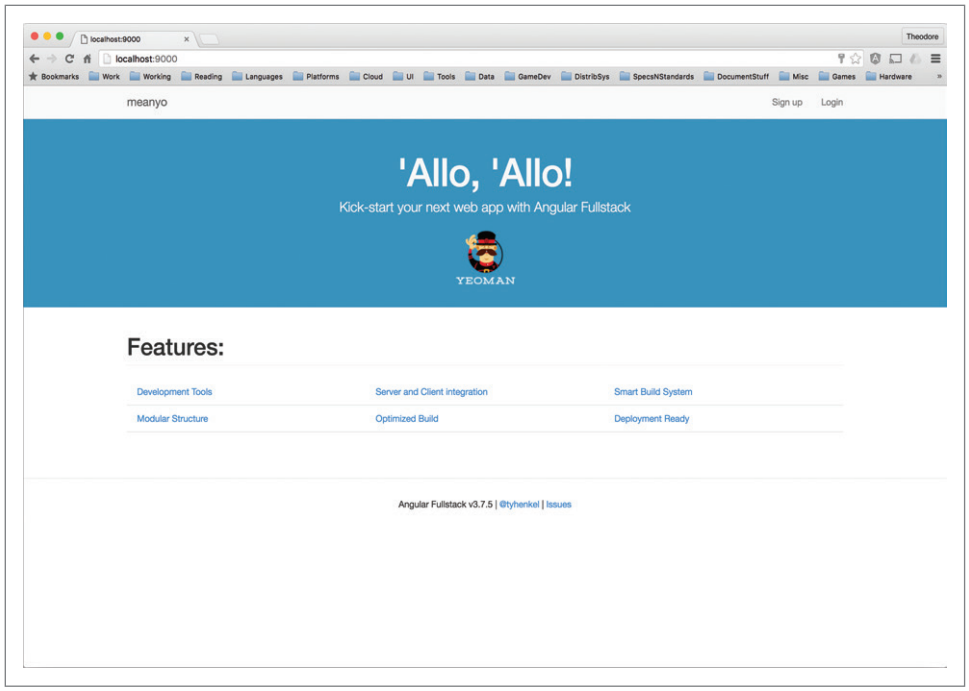


Figure 2 **The Scaffolded Application Front End**

For now, the generator you want is the "angular-fullstack" generator, because it will generate both client- and server-side scaffolding. Therefore, once you've identified which generator you want, you need to install it using the npm tool. (Yeoman doesn't use yo to manage generator installs; it relies on npm to handle that.) Thus, the next step is to "npm install --g generator-angular-fullstack." Note the "generator-" prefix; this is the convention for all Yeoman generator packages.

For now, the generator you want is the "angular-fullstack" generator, because it will generate both client- and server-side scaffolding.

Once installed, Yeoman can use it by simply referencing it (without the "generator-" prefix) as a parameter to the "yo" command: "yo angular-fullstack." At this point, Yeoman will pass control of what happens next to the generator itself, and in the case of the angular-fullstack generator, it'll begin to ask questions about what kind of application you want to scaffold out:

- whether to use Babel or TypeScript
- whether to use HTML or Jade (a popular JavaScript HTML templating library) for markup
- which CSS tool to use (raw CSS, Sass, Stylus or Less)
- which Angular router to use (ngRoute or uiRouter)
- whether to include Bootstrap
- whether to include UI-Bootstrap (an extension to Angular for Bootstrap features)
- whether to use Mongoose (which you've seen before) or Sequelize (modeling for RDBMSes) for the models in the application
- whether to scaffold out the Passport authentication code, and if so, for which services (choosing from Google, Twitter or Facebook)
- whether to use Grunt or Gulp (which I'll examine next time) as the project build tool
- finally, which testing tools to use (Jasmine or a collection of several libraries together)

Once the Q&A is done, Yeoman will generate a slew of files (most of which will fall into either the generated "client" or "server" directories it creates). It will then run an "npm install" to pull down all the dependencies for the server and run a "bower install" (which may require installing bower, "npm install –g bower") to do the same for the client. Note that because the various dependencies that are pulled in by these steps will vary depending on the exact version of the generator (and the various libraries referenced by the generated code), it's possible (likely, even) that these steps will generate warnings.

Once finished, though, you'll have a fully fleshed out scaffolded application. It won't do much, but if you kick it off with "gulp serve" as the generated README indicates (which will require having Gulp installed—"npm install –g gulp-cli"—if it's not already present), and if MongoDB is running on your local box, it'll bring up a page similar to what the ASP.NET MVC project template generates, as shown in **Figure 2**.

Note that it has full user-management support, including the ability to authenticate using Google, Facebook, or Twitter (though using each of those will require obtaining the appropriate credential tokens or keys from each of those services and specifying them in the configuration directory in the server code) or using e-mail/passwords.

Adding Speakers, Yo

Because the application you've been building (sort of) has been a speaker-rating system, one of the first things you'll need to model in this new codebase is that of a speaker. And while you could start diving into the generated files to find out where models are declared and so on, it's much easier to let Yeoman help with that: "yo angular-fullstack:endpoint speaker." It'll ask what you want to use for the API endpoint URL, and then go do its thing. This is the command-line equivalent of using Yeoman to do the ASP.NET MVC right-click/Add-Controller thing, and will generate some "emptyish" files in server/api/speaker for you to modify.

It turns out that the angular-fullstack Yeoman generator can do this for a number of different elements of the application, both client-side and server-side. You can see the full list of "sub-generators" by asking the generator itself for a list via "yo angular-fullstack --help".

Layout, Yo

Because I haven't really explored the client side of things, I'll leave that be for now. But the server side is something I've been exploring for some time, so without further ado, let's go plunging into that.

First, all the server code—not surprisingly—is contained in the "server" directory. This directory contains the main application file, app.js, the list of Express routes in routes.js and several subdirectories that contain the rest of the server-side code. That includes "api" (where the model- and controller-related code resides), "auth" (which is where the authentication code resides), "components" (which will contain non-API-related components), "config" (configuration) and "views" (which only contains a 404.html file, for when you request unknown URLs).

The auth directory is pretty much done as is; there will rarely be, if ever, a need to wander around in here. As its name implies, Config contains the configuration values used by the rest of the application in much the same way that I built up the "config.js" file in my previous columns. Most of the developer action will happen in the api subdirectory.

Figure 3 **The Speaker Mongoose Schema**

```
'use strict';

import mongoose from 'mongoose';

var SpeakerSchema = new mongoose.Schema({
  firstName: {
    type: String,
    required: true
  },
  lastName: {
    type: String,
    required: true
  },
  topics: [String],
  active: Boolean,
  created: {
    type: Date,
    default: Date.now
  },
  updated: Date
});

SpeakerSchema
  .pre('save', function(next) {
    this.updated = new Date();
    next();
  });

export default mongoose.model('Speaker', SpeakerSchema);
```

Within api, you'll find three subdirectories: the one you just created ("speaker"), one specifically for modeling users ("user") and one that's generated automatically for you by the generator ("thing") as a template or example to follow. Within each of these API subdirectories, you'll find a pattern—a collection of "double-extension" files: thing.controller.js, thing.events.js, thing.model.js and so on. There will also be an index.js file, which serves as a sort of "single entry-point" for the entire subdirectory (it brings together each of the other files together for easy reference from the rest of the directory) and an index.spec.js file, which is used specifically to test the code found in this directory.

So, for example, if you want to indicate that speakers have a first name, last name, and list of topics they like to speak on, you can open the speaker.model.js file, and model it using standard Mongoose facilities (I discussed this in an earlier column; see msdn.com/magazine/mt683801), as shown in **Figure 3**.

(Note the use of the "export"; this is a new ECMAScript 2015 feature, as discussed in my September 2015 column at msdn.com/magazine/mt422588.) However, now that you've changed the model for Speaker, you need to update the tests (in speaker.integration.js), or else tests will fail when you run them. I'll leave that as an exercise for you as a way of exploring the code; run the tests with "gulp test:server" to avoid the client-side tests. Of course, you can always explore the API by using curl ("curl localhost:3000/api/speakers," which will be empty unless you insert a few via POSTing to that endpoint or directly into MongoDB. Note that the generators are undergoing continuous development, so future versions of the generator will set a different default port or URL).

Wrapping Up

This hasn't been a particularly code-heavy column, yet you just rebooted the entire application, gained a whole ton of functionality, and essentially brought the application up to the same level (and beyond) from what you'd been building for the past year or so. Gotta love scaffolding! More important, having built all the parts piece-by-piece by hand prior to running the scaffolding, it's much easier to understand the code as a whole and what's happening where. For example, opening up routes.js will look familiar to the routing table you built by hand earlier, and the package.json (in the root of the project directory) will be bigger, though it will basically remain the same as you had been using.

The only new thing, in fact, beyond the use of Yeoman itself, is the introduction of a "build tool" to gather all the pertinent parts together into the right place, and that will be what I discuss next time. Until then, however … happy coding! ■

Ted Neward *is a Seattle-based polytechnology consultant, speaker and mentor. He has written more than 100 articles, is an F# MVP and has authored and coauthored a dozen books. Reach him at ted@tedneward.com if you're interested in having him come work with your team, or read his blog at blogs.tedneward.com.*

Thanks to the following technical expert for reviewing this article: *Shawn Wildermuth*

Windows PowerShell Just Keeps Getting Better

In a departure from my recent focus on .NET Core, in this month's Essential .NET Column I'm going to focus on a host of new features that significantly increase the power of Windows PowerShell. To me, the most significant improvements are in the area of cross-platform support—yes, really, PowerShell now runs on Linux. Not only that, but it has also moved to open source on GitHub (github.com/PowerShell/PowerShell) so that the community at large can begin to bolster its features. Cool!!

But the most recent announcements don't tell the whole story. Back in February, PowerShell 5.0 was released and it includes new or improved support for class and enum declarations, module and script discovery, package management and installation, OData endpoint access, enhanced transcription and logging, and more. In this article I'm going to review each of these features and provide examples.

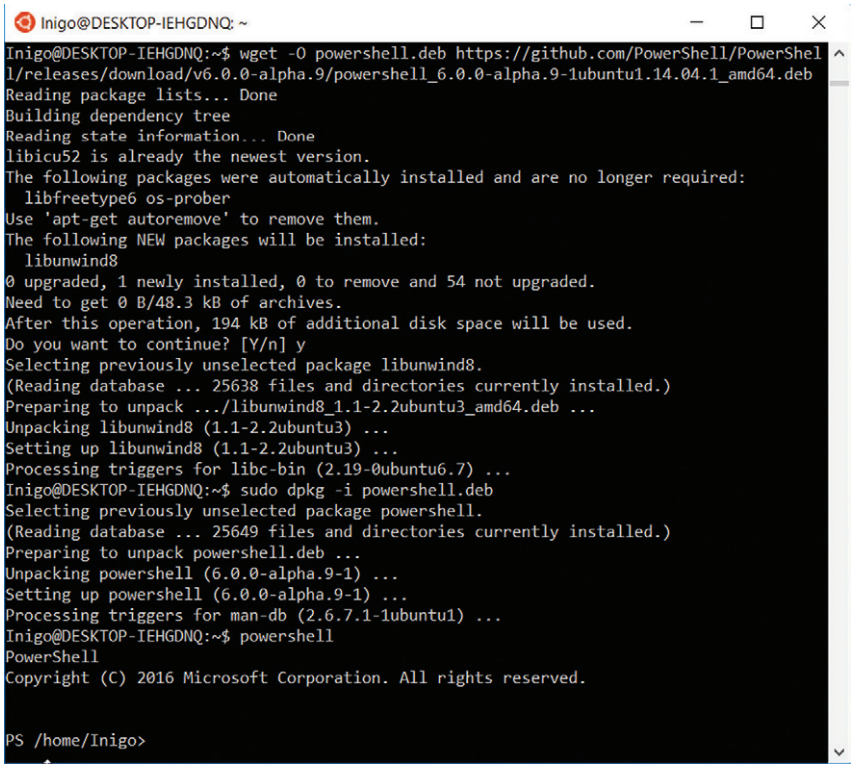


Figure 1 **Installing and Running Windows PowerShell on Ubuntu 14.04 from Bash on Ubuntu on Windows**

PowerShell Goes Cross-Platform

To begin, take a look at the following command script, which installs PowerShell on Ubuntu 14.04 from Windows PowerShell Host, along with a screenshot of the execution session from Windows Bash in **Figure 1** (For those of you not familiar with Bash running Ubuntu on Windows 10 Anniversary Update see "Installing Bash on Windows 10."):

```
wget -O powershell.deb https://github.com/PowerShell/PowerShell/releases/
download/v6.0.0-alpha.9/powershell_6.0.0-alpha.9-1ubuntu1.14.04.1_amd64.deb
sudo apt-get install libunwind8 libicu52
sudo dpkg -i powershell.deb
powershell
```

Note that the command script specifically targets Ubuntu 14.04. For other platforms, the deb package URL and the prerequisite versions will vary. See bit.ly/2bjAJ3H for instructions on your specific platform.

Many years ago now Jeffrey Snover tweeted that PowerShell could reasonably be expected to appear on Linux, but it has taken so long and there have been so few progress reports that even today as I use it I'm amazed. I mean, really? I'm running Bash on top of Ubuntu running on Windows (without leveraging any virtualization technology) and (assuming I don't want to install PowerShell directly into the same Bash instance) using SSH to connect to a remote Bash session where I can install PowerShell and pipe .NET objects between commands within the Bash shell.

If I had suggested this would be possible a couple of years ago I doubt many would have believed me.

PowerShell Repositories and the PowerShell Gallery

While it's great that you can write your own scripts and libraries, it's likely that someone else in the community has already done something similar that you can leverage and improve upon. Until the advent of the PowerShell Gallery (PowerShellGallery.com), however, you had to comb the Internet to find scripts and modules that might be useful—whether they were community contributions or official PowerShell product releases like Pscx or the Posh-Git module. One of the more recent PowerShell improvements (part of PowerShell 5.0) I've become completely dependent on is the new repository support,

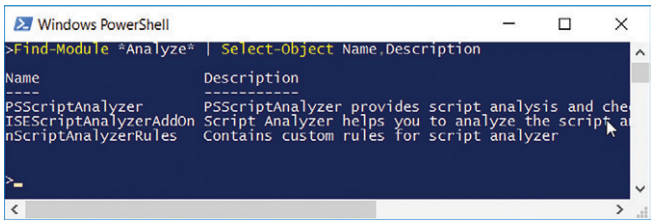


Figure 2 **Output of Find-Module Command**

specifically the PowerShell Gallery. Imagine, for example, that you've been writing PowerShell for some time and, in so doing, you've become aware that there are many pitfalls to be avoided, if only there was a way to analyze your code and find them. With this in mind, you could browse to the PowerShell Gallery and search for an analyze module to install. Or, even better (because you presumably already have a PowerShell window open), you can leverage the PowerShellGet module's Find-Module command (included with PowerShell 5.0):

```
Find-Module *Analyze* | Select-Object Name,Description
```

The output of which is shown in **Figure 2**.

Note that if you don't have a sufficiently modern version of NuGet installed, leveraging the PowerShellGet module will trigger a NuGet package update.

Assuming you find the module you want, you can view its contents via the Save-Module command. To install the module, use the Install-Module (in this case, Install-Module PSScriptAnalyzer) command. This will download the module and install it for you, making all the functions included in the module available. After installing the PSScriptAnalyzer module, you can invoke Invoke-ScriptAnalyzer $profile to scan your profile and identify concerns that the analyzer considers suboptimal. (Note that it's no longer necessary to import a module in order to access it. Module functions are automatically indexed such that when you invoke a module function, the module will automatically import and be accessible on demand.)

Note that the PowerShell Gallery is configured as a repository by default:

```
>Get-PSRepository

Name         InstallationPolicy   SourceLocation
----         ------------------   --------------
PSGallery    Untrusted            https://www.powershellgallery.com/api/v2/
```

As a result, Find-Module works without issue. However, Install-Module will prompt you with an untrusted repository warning. To avoid this, assuming you do indeed trust the repository, you can set it to trusted with the command:

```
Set-PSRepository -Name PSGallery -InstallationPolicy Trusted
```

Installing Bash on Windows 10

Starting with Windows 10 Anniversary Edition, you can install Bash natively onto Windows with the following Windows PowerShell command:

```
Get-WindowsOptionalFeature -Online -FeatureName *linux* | Enable-
WindowsOptionalFeature -NoRestart -all -online
```

Note, however, that this is still in beta and, therefore, only enabled in developer mode (use "Get-Help WindowsDeveloper" to see how to explore developer mode).

Unfortunately, this feature does require a restart, but I include the -NoRestart option so that enabling the feature doesn't directly trigger the restart.

Apt-Get for Windows with PowerShell Package Management

Those of you who have spent any time as an IT professional in the Linux world have no doubt taken apt-get for granted—likely with install scripts that bootstrap their environment the moment they start up a new Linux instance. For those of you who haven't, apt-get is a command-line way to download and install programs/packages and any dependencies quickly and easily from the Internet right from the command line. **Figure 1** shows a trivial example of such an installation when it leverages apt-get to install libunwind8 libicu52, on which PowerShell (on Ubuntu 14.04) depends. With PowerShell 5.0, the same functionality comes to Windows (I'm not sure whether to shout, "Yahoo!" or exasperatedly sigh, "Finally!"—perhaps both).

> To me, the most significant improvements are in the area of cross-platform support—yes, really, PowerShell now runs on Linux.

Just as there are repositories, like PowerShell Gallery, for PowerShell modules, PowerShell 5.0 also includes support for managing programs—called packages—in Windows. One such package manager is Chocolatey (chocolatey.org) and you can add it as a package repository using the following command:

```
Get-PackageProvider -Name chocolatey
```

This allows you to use PowerShell to find packages that have been deployed into Chocolatey. For example, if you want to install Visual Studio Code all you have to do is enter the commands:

```
Find-Package V*S*Code | Install-Package
```

As shown, wild cards are supported.

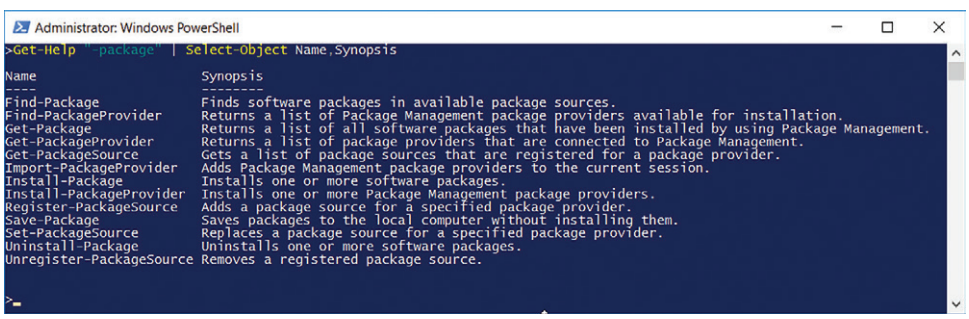


Figure 3 **Available Windows PowerShell Package Commands**

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

Other Package commands to be familiar with are available using the following command, with the results shown in **Figure 3**:

```
Get-Help "-package" | Select-Object Name,Synopsis
```

As you can see, you can both get and uninstall a package. Get-Package lists all the programs (and more) available from the Control Panel Programs and Features. Therefore, if you wanted to uninstall Notepad2, for example, you could use the command:

```
Get-Package Notepad2* | Uninstall-Package
```

The ease this brings to automating Windows computer setup is tremendous. I've been a Chocolatey fan for a number of years now and this integrates Chocolatey support directly into Windows. It ultimately brings package management to Windows in much the same way that Apt-Get does on Linux.

One thing to consider is that not only can the Chocolatey repository be accessed via the *-package* PowerShell commands, but Chocolatey can also be installed directly. While not required, installing Chocolatey directly will occasionally provide a more robust feature set of package management functionality. Fortunately (and perhaps ironically), installing Chocolatey is simply a matter of invoking Install-Package Chocolatey, but (and this is an example of the discrepancies between Chocolatey and *-Package behavior) the default install location will depend on which installation engine is used. Check out chocolatey.org/install for more information on the Chocolatey toolset, including installation instructions for your environment.

OData with Export-ODataEndpointProxy

Another PowerShell 5.0 feature that's worth mentioning is the ability to generate a set of methods that access an OData data source such as Visual Studio Team Services (VSTS). **Figure 4** demonstrates running the Export-ODataEndpointProxy on an OData service, a public sample Northwind OData service in this case.

If you browse the generated module commands, you'll notice that separate commands are generated for each entity (Advertisement, Category, Person and so forth), along with corresponding actions for each (Get, New, Remove, Set).

One thing to note on the command line in **Figure 4** is the use of the -AllowUnsecureConnection parameter. This is necessary because the OData service used in this example doesn't require authentication or encryption.

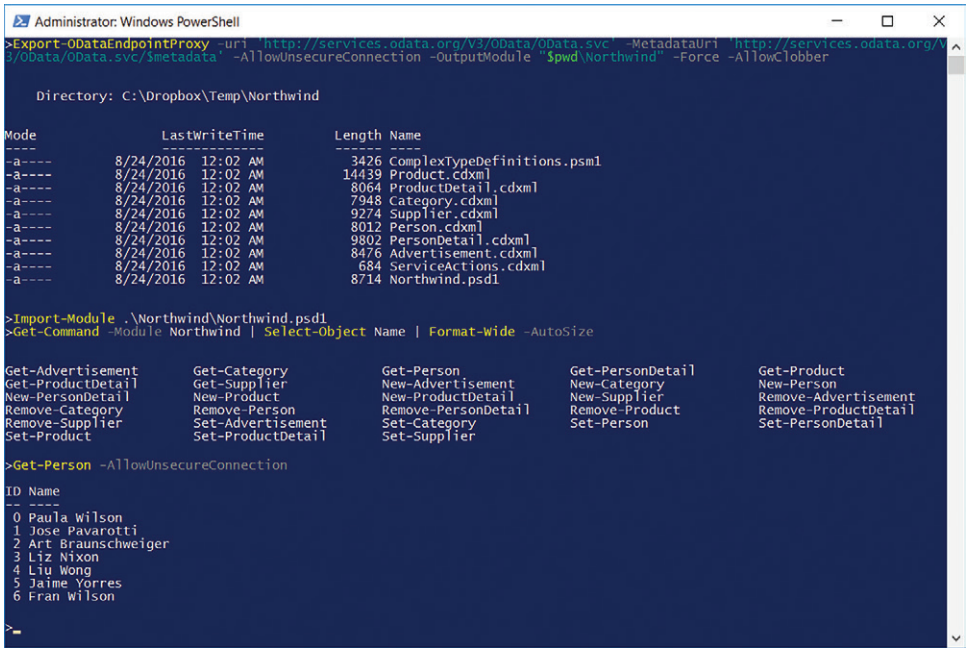


Figure 4 **Generating and Invoking an OData Proxy**

Converting from Text to Objects with ConvertFrom-String

Another new command to appear in PowerShell 5.0 is Convert-From-String. It's designed to take structured text as input and interpolate the structure so as to output an object based on the parsed text. You could use this, for example, to parse a flat file or (and this is where I find it extremely useful) to convert the text output from an executable into an object.

Consider, for example, SysInternal's handle.exe program, (which you can install using the Install-Package Handle command—leveraging package management as discussed in the previous section). As you'd expect from a command-line utility, it writes out text to stdout—in this case a list of open handles associated with a name. In PowerShell, however, you've grown accustomed to

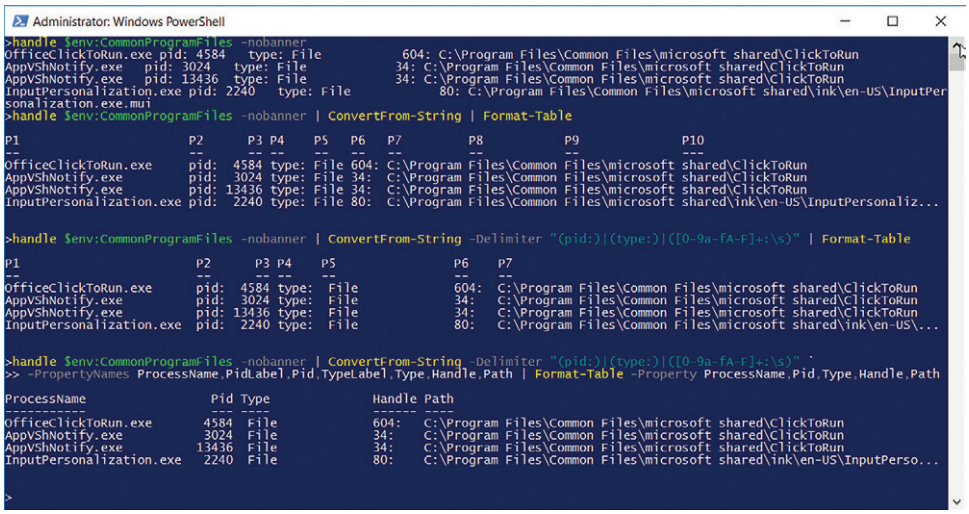


Figure 5 **Utilizing ConvertFrom-String to Parse stdout into an Object**

working with objects. And, to convert the text output into an object, you use the ConvertFrom-String function, as shown in **Figure 5**.

Figure 5 starts by showing the raw output of the handle.exe utility. Next, it demonstrates ConvertFrom-String without any parameters. As a result, the ConvertFrom-String utility simply splits the text on each line based on white space.

In the third example, I demonstrate the option of specifying a regular expression split pattern in order to fine-tune the parsing. However, note that familiarity with regular expressions isn't required. You can instead specify a template—perhaps more accurately a sample—of either a file or a string, in which you parse the first few items manually. ConvertFrom-String then leverages the sample-parsed content and interprets how to parse the remainder of the input.

In the final example, I added the -PropertyNames parameter so as to assign meaningful names to the output.

In the end, ConvertFrom-String bridges the impedance mismatch of the text-based world of the traditional process stdout with a PowerShell world built on objects. In this case, I can pipe the output into Stop-Process -Id mapping the pid value into the -Id parameter value.

Classes and Enums

Finally, here's a rundown on the new class and enumeration support. In PowerShell 5.0, two new keywords were added corresponding to the two structures so that you can now declare a class or an enumeration directly in PowerShell (rather than using Add-Type and passing C# code or perhaps instantiating a PSCustomObject). The syntax is what you'd expect—see **Figure 6**.

Notice in particular, that both properties and methods are supported. Furthermore, there are declaration modifiers like static and hidden, which designate the associated construct accordingly. Furthermore, inheritance is supported with a syntax very similar to C#:

```
class Employee : Person {}
```

Figure 6 **Declaring Classes and Enums in Windows PowerShell**

```
enum CustomProcessType {
  File
  Section
}
class CustomProcess {
  [string]$ProcessName;
  hidden [string]$PIDLabel;
  [int]$PID;
  hidden [string]$TypeLabel;
  [CustomProcessType]$Type;
  [int]$Handle;
  [string]$Path;
  CustomProcess(
    [string]$processName,[string]$pidLabel,[int]$pid,
    [string]$typeLabel,[string]$type,[int]$handle,[string]$path) {
    $this.ProcessName = $processName;
    $this.PIDLabel=$pidLabel;
    $this.PID=$pid;
    $this.TypeLabel=$typeLabel;
    $this.Type=$type;
    $this.Handle=$handle;
    $this.Path=$path;
  }
  CustomProcess() {}
  GetProcess() {
    Get-Process -Id $this.PID
  }
  static StopProcess([CustomProcess]$process) {
    Stop-Process -Id $process.PID
  }
}
```

Last, and also demonstrated in **Figure 6**, constructors can be declared. In this example, I declare a default constructor (with no parameters) and a second constructor that takes all the parameters. The constructors are invoked via the New-Object command by specifying either the -ArgumentList parameter (where an array of constructor arguments is listed) or else a HashTable argument is passed via the -Property parameter.

In the end, ConvertFrom-String bridges the impedance mismatch of the text-based world of the traditional process stdout with a PowerShell world built on objects.

Wrapping Up

By no means is this a complete list of new features in PowerShell 5.0. Other notable items include:

- Integration of archive (.zip file support) through the Compress-Archive and Expand-Archive commands.
- Get-Clipboard and Set-Clipboard commands that also work with the pipe operator.
- Out-File, Add-Content and Set-Content include a –NoNewline parameter, allowing for file content that omits the new-line character.
- The New-TemporaryFile command works similar to [System.IO.Path]::GetTempFileName (though not identically). Like its .NET equivalent, New-TemporaryFile doesn't delete the temporary file, so be sure to save the output so you can remove the file once you're done with it.
- SymbolicLinks can now be managed directly from the PowerShell cmdlets New-Item and Remove-Item.
- PowerShell Integrated Scripting Environment (ISE) now supports logging via the Start/Stop/Search-Transcript functions, which previously errored when called from PowerShell ISE.

Furthermore, while not supported immediately in the open source release of PowerShell, Microsoft fully intends to support Open SSH such that it will be a remoting transport option in PowerShell, as well as in Windows Remote Management, not long after this article is published.

All this to say, PowerShell just keeps getting better and better. If you haven't learned it yet, get going. ■

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# 6.0 (5th Edition)." Contact him on Twitter: @markmichaelis or via e-mail at mark@IntelliTect.com.*

Thanks to the following IntelliTect technical experts for reviewing this article: *Kevin Bost, Phil Spokas and Michael Stokesbary*

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

LAS VEGAS
MARCH 13-17 2017
BALLY'S, LAS VEGAS, NV

EVENT PARTNERS

Microsoft Magenic

SUPPORTED BY

Visual Studio msdn magazine Visual Studio MAGAZINE

PRODUCED BY

1105 MEDIA

INTENSE TRAINING FOR DEVELOPERS, ENGINEERS, PROGRAMMERS, ARCHITECTS AND MORE!

Track Topics include:

- Visual Studio / .NET Framework
- JavaScript / HTML5 Client
- Modern App Development
- Mobile Client
- Software Practices
- Database and Analytics
- Angular JS
- ASP.NET / Web Server
- Agile
- ALM / DevOps
- Cloud Computing
- Windows Client

Register NOW and Save $500!

Use promo code VSLOCT2 Scan the QR code to register or for more event details.

ROCK YOUR CODE TOUR 2017

LAS VEGAS	AUSTIN	WASH. DC	REDMOND	CHICAGO	ANAHEIM	ORLANDO
MARCH 13-17	MAY 15-18	JUNE 12-15	AUG 14-18	SEPT 18-21	OCT 16-19	NOV 13-17

vslive.com/lasvegas

CONNECT WITH US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

DAVID S. PLATT

A Technical Solution to a Political Problem

I'm sick of this U.S. presidential election cycle. I was sick of this election cycle a year ago. As I write these words in August, I despair at the thought of 100 more days of this ever-increasing cacophony.

I've never been a fan of politics, but this election is especially bad. As Ian Tuttle wrote in early August (bit.ly/2bkAt2T), we have "two small groups of extreme partisans fighting on behalf of horrible candidates, and a sea of voters in between disheartened by two miserable options." Sometimes I think both sides are trying to lose (shades of "The Producers," anyone?), and wish there was some way that could happen. Maybe it's confirmation bias, but I'm noticing a lot more bumper stickers like the one in **Figure 1**.

The Web is often lauded as a medium for the exchange of ideas. That's not happening now. The volume of speech is huge, but the content is minimal. I'm omitting no detail whatsoever when I paraphrase the conversation thusly: one side shouts, "X good, Y bad," while their opponents riposte with the witty and thoughtful, "No, you idiot, Y good, X bad."

My political philosophy is simple: I agree with Mark Twain, that politicians should be changed as often as diapers, and for the same reason. Throw the rascals out, and put our rascals in. But for Pete's sake, can't we finish the damn throwing so we can get on with our lives? Especially because I've already voted by absentee ballot, and couldn't change it if I wanted to.

We geeks could provide an option for those who want out of it. I'll bet many people do by now. As Mr. Peabody always said to Sherman, "As usual, I have a plan."

Most people I know use ad blockers on their Web browsers. I'm sure that the lack of one hampered the adoption rate of the Microsoft Edge browser. (It just got one over the summer.) How hard would it be to concoct a blocker for political content? I think not very.

Figure 1 **A 2016 Presidential Election Bumper Sticker**

Content blockers were the second mass-market product to appear on the consumer Internet. Parents bought them to shield their children from pornography (the first mass-market product), and their capabilities have expanded over the years. Net Nanny, the market leader, advertises that it can filter 18 categories of content, from dating to alcohol to abortion. The primary limitation has always been that the kids are more computer-savvy than the parents, so the filter settings don't last long. We won't have this problem as we build our content blockers for consenting adults.

The early content blockers were blunt. But today's Net Nanny claims to be smart enough to distinguish the word "breast" in a cooking context (chicken), a medical context (cancer) or a sexual context (augmentation). Surely such an engine could distinguish between Trump the candidate and trump in a card game. I'll call it Plattski's Political Pablum Preventer (P^4).

How would such a thing improve our lives? For a quick test, I scanned the front page of *The New York Times* Web site, imagining blank spaces replacing the political articles, as Adblock Plus replaces the ads with blank spaces. I'd still see all the sports scores, the technology section and daily news coverage. Just no Donald and no Hillary. Looks fine to me. Let's get it done.

I cannot close this column without renewing my eternal election-year call: Lie to the exit pollsters. The control they exercise over our society is revolting, but it's also easy to disrupt. All we have to do is lie. If you voted for A, say you voted for B, and vice versa. If you made up your mind a long time ago, tell them you just made it up in the voting booth, or the other way around. If they ask your age, add or subtract five years, whichever you think you can get away with. If they ask your gender, you'd probably better tell the truth; it might be a control question. If everyone does this, we'll have a delightful election evening of watching the prognosticators fall on their faces—the funniest night of political foolishness since Henry Kissinger won the Nobel Peace Prize. Now let's go to it. ■

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.*

GROUPDOCS
Document Manipulation APIs

Manipulating Files?

View, annotate, compare, convert, assemble, sign and share over **50 types of documents on the web.**

GroupDocs.Viewer
GroupDocs.Annotation
GroupDocs.Conversion
GroupDocs.Comparison
GroupDocs.Signature
GroupDocs.Assembly
GroupDocs.Metadata
GroupDocs.Search

Get started now

.NET Libraries | Java Libraries | Cloud APIs | Cloud Apps

Contact Us:
US: +1 903 306 1676
EU: +44 141 628 8900
AU: +61 2 8006 6987
sales@groupdocs.com

Visit us at www.groupdocs.com