THE MICROSOFT JOURNAL FOR DEVELOPERS

DECEMBER 2016 VOL 31 NO 13

msdn magazine

DevExpress

Your Next Great Desktop App Starts Here

From apps that replicate the look and feel of Microsoft Office® to high-powered data mining and decision support systems for your enterprise, DevExpress desktop controls will help you build your best, without limits or compromise.

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

devexpress.com/try

THE MICROSOFT JOURNAL FOR DEVELOPERS

DECEMBER 2016 VOL 31 NO 13

msdn magazine

Generate Code with the .NET Compiler Platform.....40

CONNECT(); SPECIAL SECTION

COLUMNS

Microsoft

WE'RE CHANGING THE WAY YOU LOOK AT REPORTING

The Text Control Reporting Framework combines powerful reporting features with an easy-to-use, MS Word compatible word processor.

Our award-winning developer libraries are completely independent from MS Word or any other third-party application and can be completely integrated into your business application.

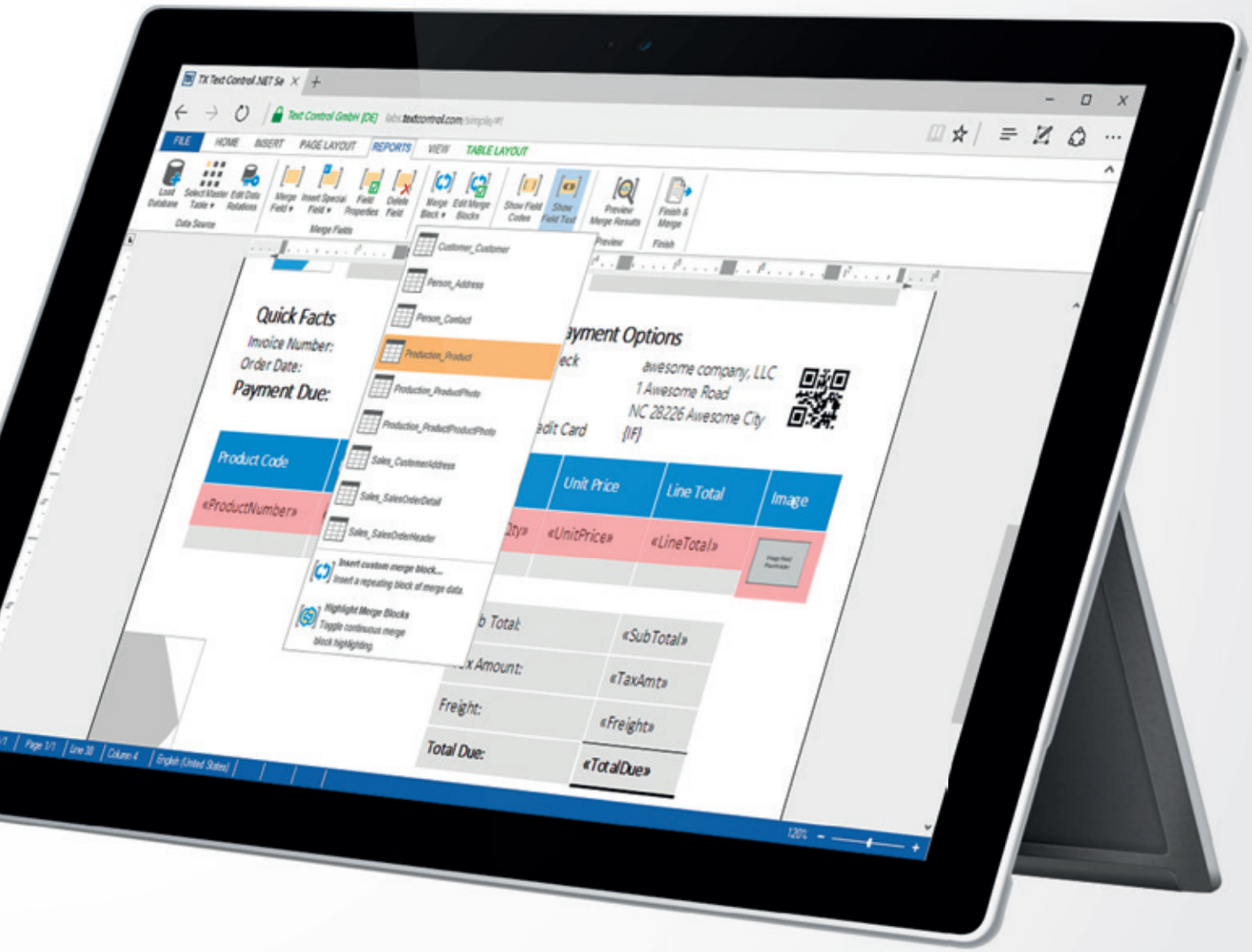
Follow the trend and switch to flow type layout reporting.

www.textcontrol.com www.reporting.cloud

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

Visual Studio Partner

REPORTING LIBRARIES FOR WINDOWS, WEB, MOBILE AND CLOUD APPLICATIONS



TEXT CONTROL

msdn magazine

DECEMBER 2016 VOLUME 31 NUMBER 13

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

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

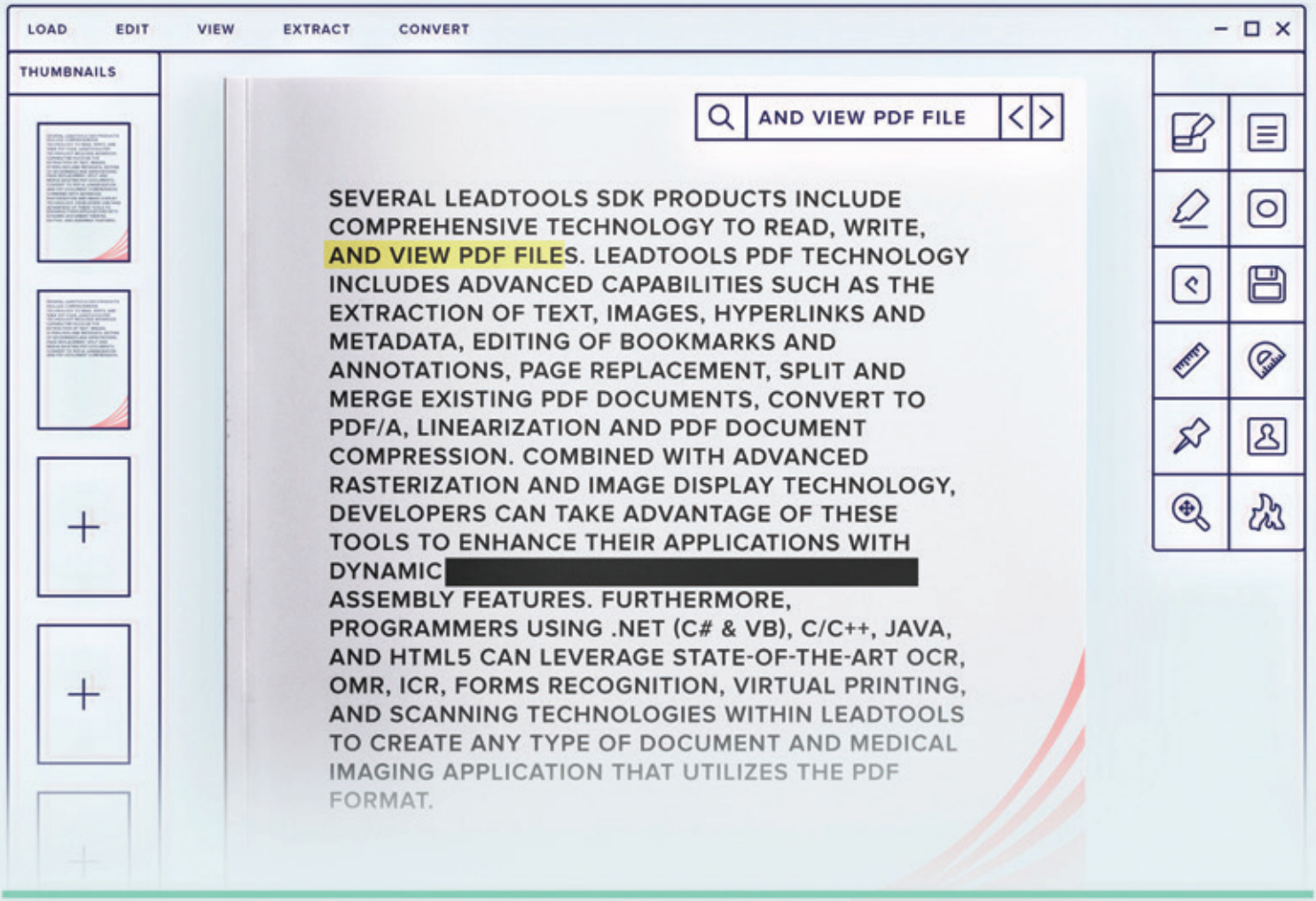
The opinions expressed within the articles and other contentsherein do not necessarily express those of the publisher.

Microsoft

LEADTOOLS®

THE WORLD LEADER IN IMAGING SDKs

PREMIER PDF SDK



Comprehensive PDF SDK technology for today's digital world. Develop multi-platform applications with support for extracting, editing, and writing text, hyperlinks, bookmarks, digital signatures, PDF forms, annotations, metadata, and pages from PDF files on any desktop, server, and mobile device.

.NET Windows API Java WinRT Linux iOS macOS Android JavaScript

DOWNLOAD OUR 60 DAY EVALUATION
WWW.LEADTOOLS.COM

LEAD TECHNOLOGIES INCORPORATED

SALES@LEADTOOLS.COM
800.637.1840

Editor's Note

MICHAEL DESMOND

The Internet of Pwned Things

In the September issue of *MSDN Magazine* last year, I wrote about a pair of hackers who demonstrated how they could seize control of a Jeep Cherokee SUV over the Internet, cracking the car's infotainment system to manipulate brakes, transmission and other controls ("The Internet of Car Wrecks," msdn.com/magazine/mt422336). As I wrote at the time:

"I see all this as an early test of the Internet of Things (IoT) concept. Carmakers are not uniquely negligent in securing IoT systems—by most accounts, almost everyone is bad at it—but the risk they face is severe."

I didn't know the half of it.

> Since the turn of the century hackers have been leveraging infected client PCs and servers to act as packet-hurling zombie armies that overwhelm targeted Web sites and services.

On a quiet Friday in October, millions of webcams, digital video recorders (DVRs) and other connected consumer devices rose up to mount a sustained distributed denial of service (DDoS) attack against Domain Name Server (DNS) provider Dyn. The digital assault hauled down the Dyn DNS service and denied access for millions of people to prominent Web sites and services, including Twitter, Tumblr, Amazon and Netflix.

Of course, DDoS attacks are nothing new. Since the turn of the century hackers have been leveraging infected client PCs and servers to act as packet-hurling zombie armies that overwhelm targeted Web sites and services. This time the attack came not from infected computers but from a vast pool of compromised Internet-connected devices. The resulting assault was unprecedented in both size and character, overwhelming the Dyn infrastructure and causing its DNS service to fall offline.

Security experts have been warning for years about the proliferation of unsecured or lightly secured devices as part of the IoT movement, and now the chickens are coming home to roost. The attack, executed using Mirai malware, involved tens of millions of IP addresses to become the most massive DDoS campaign in history, outpacing the 620 Gbps attack aimed at the KrebsOnSecurity site in September.

How were hackers able to seize control of such a vast army of IoT devices, you ask? By using the sophisticated technique of attacking the devices with their default passwords.

I hope you're detecting the sarcasm here, because I'm laying it on pretty thick.

The IoT promises to usher in a golden age of intelligence, automation and control, where millions of smart devices communicating and working in concert promise to transform everything. But the failure to secure these devices, or to establish an industry standard to set a benchmark for securing them, is producing some grim outcomes. Call it the Internet of Pwned Things.

The sad fact is, we had a chance early in this revolution to do the right thing. To take security seriously and establish a hardened fabric of connected devices that could, at the very least, resist the inevitable forces that would seek to exploit them. But we didn't do that. Even carmakers, whose products could place their customers in mortal peril, failed to take rudimentary steps to secure their devices.

But on a quiet Friday in October, everything changed. And maybe now we will see meaningful action to secure and protect the devices that we expect to shape the digital landscape for decades to come.

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

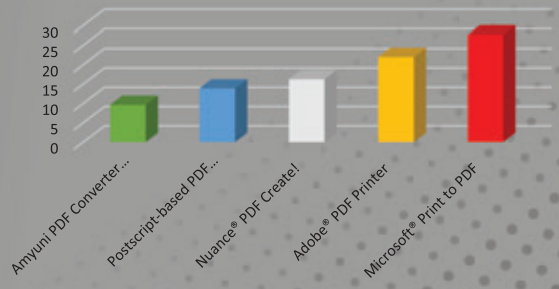
Other Developer Components from Amyuni®

- 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

Big or Small

The choice between working for a startup and working for a larger company isn't always simple. I've worked at tiny startups and massive tech companies, and I've switched back and forth, many times. When deciding among job offers, I find these seven questions helpful in making the right decision:

Do I want stack breadth or stack depth? At a startup, often, the company's growth may be so rapid, the competitors and industries moving so quickly, that my responsibilities and projects change along with it. This lets me experience more of the stack, even if I'm a back-end engineer.

On the other hand, at a larger company, I'm often part of a multi-year product or division-wide strategy that's pre-decided. This inherent stability allows me to go more in-depth into not just one part of the product, but also one part of the stack. Instead of being a back-end engineer, I could be the back-end engineer who specializes in data serialization.

> At a big company, if I have a bad manager or particularly terrible coworkers, I can switch to a different team.

Do I want to explore my career interests, or am I set on what I want? If I'm 100 percent set on being a software engineer and enjoy that role, then the sheer variety and overwhelming technical problems inherent in working for a startup keep me entertained. Yet, maybe I'm not so set on engineering. Maybe I want to try being a product manager (PM), marketer, financial manager or salesperson. Maybe I just want to jump from one discipline to another every few years, and still be safe if one of those roles goes awry.

If I don't know what I want—or if sheer variety is what I want—then large companies are the perfect playgrounds. I can be an engineer for two years, then a PM for two years, then a technical writer for two years, a developer advocate for two years, a marketer for two years. When I make such switches within one company, my varied experiences keep my career trajectory moving upward! I don't sacrifice career velocity with experimentation.

What's my exit strategy if I work with unbearably difficult people? At a big company, if I have a bad manager or particularly terrible coworkers, I can switch to a different team. Yet, at a smaller startup, I could be "stuck" under a tyrannical boss, a moody cofounder, lower-caliber coworkers, or an imploding company or industry. The only escape route is to switch to a different startup. There's no guarantee how bearable the new place will be.

While big companies have all these same problems, too, it can be psychologically less taxing to switch teams within a larger company.

Do I want to be a manager or an individual contributor? Startups offer more opportunities to move up the management chain due to more employee churn—loyalties are inherently less strong, lifers less common, and if the company does well, expansions and promotions more rapid. At bigger companies, I may have to wait for my manager's untimely demise to move up.

Do I relish older coworkers or younger ones? Regardless of what my own age is, if I feel more alive around a vibrant, younger crowd or feel young at heart, I gravitate toward startups. (How to combat perceived ageism is its own topic; contact me for my strategies around that.) If I feel more at home around more staid colleagues, I pick bigger companies or startups with a mixed-age profile.

Do I have my act together, or am I still learning to be productive? Perhaps, currently, I waste all my time on Facebook at work. Or, perhaps, I'm very inconsistent—productive one month and super-lazy the next. Or, maybe, I have anger issues that result in me blowing up regularly at coworkers!

Whatever the issue, at bigger companies, I'm less likely to be fired. My behavior is less in the spotlight, has less direct impact on the company or its culture, and so I have more leeway. This safety gives me a container to work through and heal my issues. That way, if a startup is what I want, I can make the switch once I heal my wounds.

Ultimately, it's folly to think that switching companies will solve deep emotional or psychological problems. If a change in external environment appears to solve internal issues immediately, it's often temporary. Especially at a startup, if my issue is glaring enough, I'll be shown the door quickly, with little mercy.

Am I optimizing for my career, or am I optimizing for my life? If I have a health issue, if I have small kids, if I'm taking care of family members, if I want to start a side business or explore hobbies or interests, a bigger company *usually* (but not always!) makes it easier.

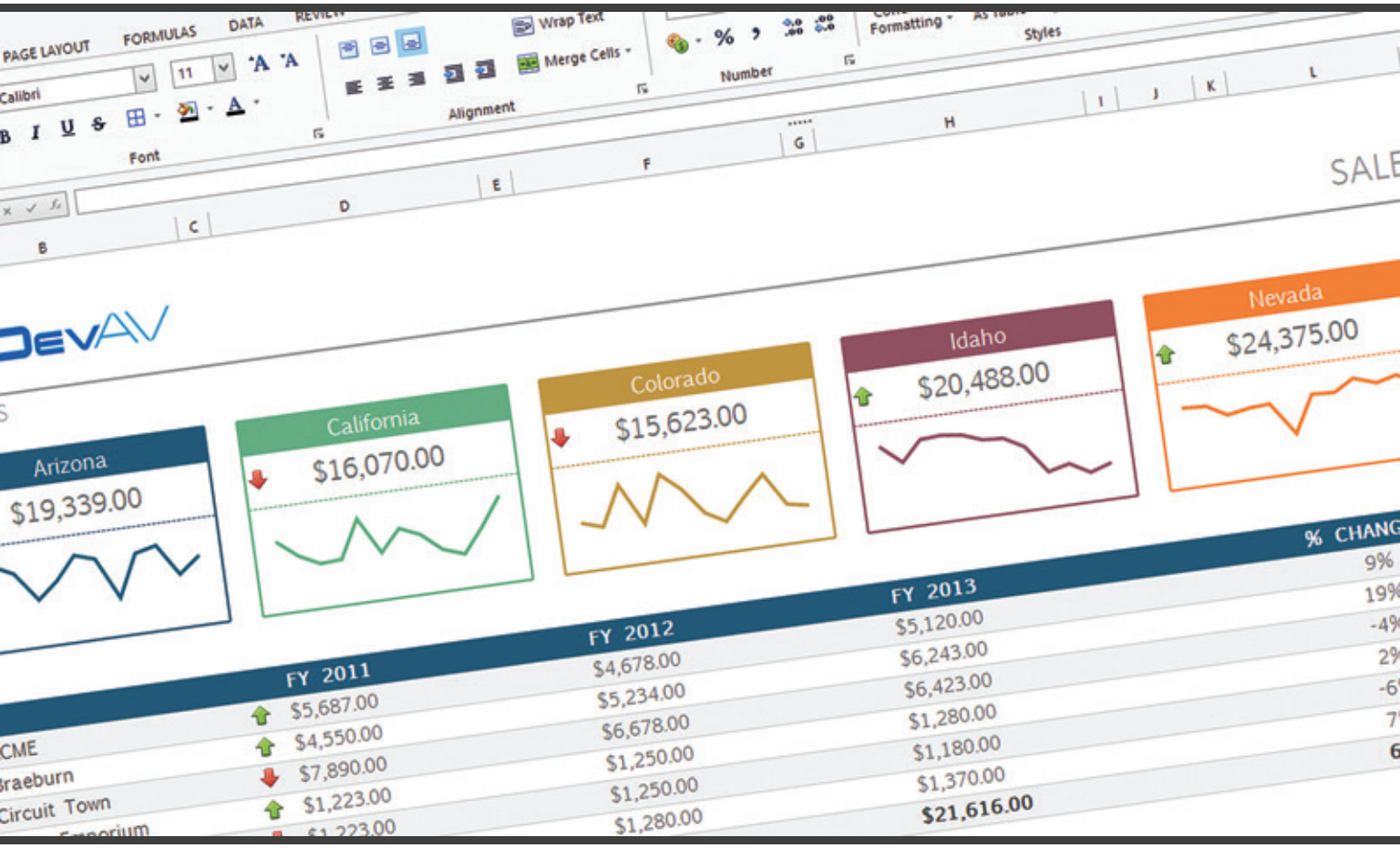
If those aren't considerations, and I want to feel more alive at work, if I find that a more active career enlivens the rest of my life, and if I can spend more time at work or see work as an extension of my social life, a startup is usually a better choice. ■

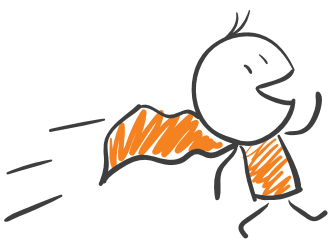
KRISHNAN RANGACHARI *is a career coach for software engineers. Visit RadicalShifts.com for his free courses for developers.*

DevExpress®

Office® Inspired Apps

Get started today and create high-performance, high-impact .NET solutions that fully replicate the look, feel and user-experience of Microsoft Office.®





Your Next Great Business App Starts Here

Explore our complete range of Office-inspired controls for all major .NET platforms.
devexpress.com/office

DINO ESPOSITO

Rewrite a CRUD System with Events and CQRS

The world's full of classic Create, Read, Update, Delete (CRUD) systems built around a relational database and padded with chunks of business logic, sometimes buried in stored procedures and sometimes caged in black box components. At the core of such black boxes are the four operations of CRUD: creating new entities, reading, updating and deleting. At a sufficiently high level of abstraction, that's all of it—any system is, to some extent, a CRUD system. The entities might be quite complex at times and take more the form of an aggregate.

In Domain-Driven Design (DDD), an aggregate is a business-related cluster of entities with a root object. Hence, creating, updating or even deleting an entity might be subject to several intricate business rules. Even reading the state of an aggregate is usually problematic, mostly for UX reasons. The model that works to alter the state of the system isn't necessarily the same model that works for presenting data to users in all use cases.

Taking the abstraction level of CRUD to its highest leads straight to separating what alters the state of a system from what simply returns one or more views of it. This is the raw essence of Command and Query Responsibility Segregation (CQRS), namely the neat segregation of command and query responsibilities.

However, there's more to it that software architects and developers must take into account. The state of the system is altered at the command stack and, in concrete terms, that's where aggregates are first created; that's also where the same aggregates are later updated and deleted. And that's precisely the point to rethink.

Preserving history is critical for almost any software system. As software is written to support ongoing business, learning from the past is crucial for two reasons: to avoid missing a single thing that happened and to improve services to customers and employees.

In the May 2016 (msdn.com/magazine/mt703431) and June 2016 (msdn.com/magazine/mt707524) installments of this column, I presented ways to extend the classic CRUD to a historical CRUD. In my August 2016 (msdn.com/magazine/mt767692) and October 2016 (msdn.com/magazine/mt742866) columns, instead, I presented an Event-Command-Saga (ECS) pattern and a Memento FX framework (bit.ly/2dt6PVD) as the building blocks of a new way to express the business logic that meets everyday needs.

In this column and the next, I'll address the two aforementioned benefits of preserving history in a system by rewriting a booking demo application (the same I used in the May and June columns) with CQRS and event sourcing.

Code download available at bit.ly/2frqwU2.

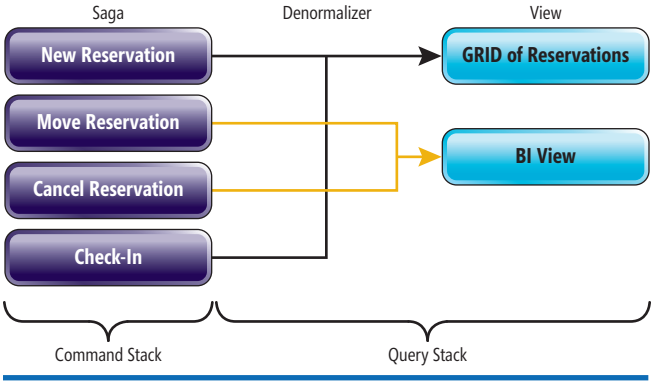


Figure 1 **User Actions and High-Level Design of the System**

The Big Picture

My sample application is an internal booking system for meeting rooms. The main use case is a logged user that scrolls a calendar and books one or more free slots on a given room. The system manages entities such as Room, RoomConfiguration and Booking, and, as you can imagine, conceptually the whole application is about adding and editing rooms and configurations (that is, when the room is open for booking and length of single slots), and adding, updating, and canceling reservations. **Figure 1** offers a glimpse of the actions that users of the system are able to perform and how they'll be architected in a CQRS system according to the ECS pattern.

A user can enter a new reservation, move and cancel it, and even check into the room so that the system knows the reserved room is actually being used. The workflow behind each action is handled in a saga and the saga is a class defined in the command stack. A saga class is made up of handler methods, each processing a command or an event. Placing a reservation (or moving an existing reservation) is a matter of pushing a command to the command stack. Generally speaking, pushing a command can be as simple as directly invoking the corresponding saga method or it can go through the services of a bus.

To preserve history, you need to track at least all the business effects of any processed commands. In some cases, you also might want to track the original commands. A command is a data-transfer object carrying some input data. A business effect of executing a command through a saga is an event. An event is a data-transfer object carrying the data that fully describes the event. Events are saved to a specific

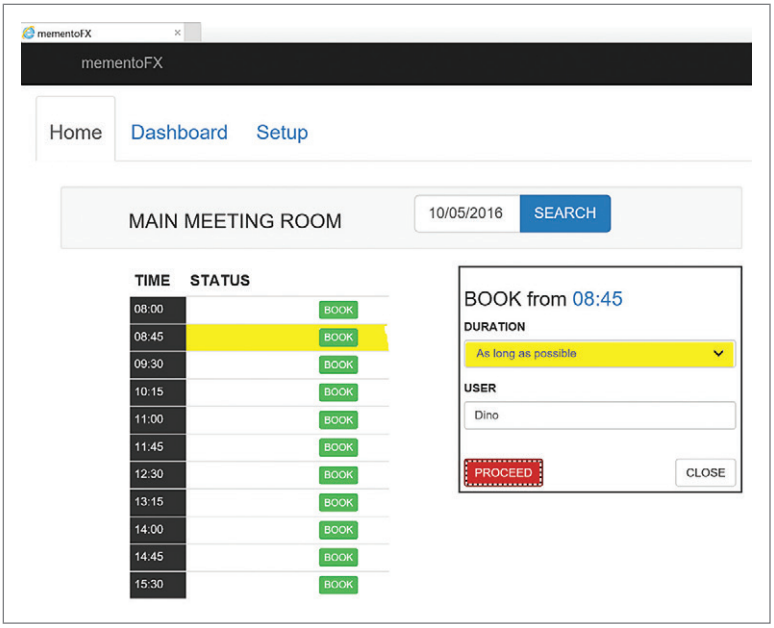


Figure 2 **Booking a Meeting Room in the Sample System**

data store. There are no strict constraints on the storage technology to use for events. It can be a plain relational database management system (RDBMS) or it can be a NoSQL data store. (Refer to the October column for the setup of MementoFX and RavenDB and bus.)

Coordinating Commands and Queries

Let's say that a user places a command to book a slot on a given room. In an ASP.NET MVC scenario, the controller gets the posted data and places a command to the bus. The bus is configured to recognize a few sagas and each saga declares the commands (and/or events) it's interested in to handle. Hence, the bus dispatches the message to the saga. The input of the saga is the raw data that users typed in the UI forms. The saga handler is responsible for turning received data in an instance of an aggregate that's consistent with the business logic.

> To preserve history, you need to track at least all the business effects of any processed commands.

Let's say that the user clicks to book, as shown in **Figure 2**. The controller method triggered by the button receives the ID of the room, the day and time, and the user name. The saga handler must turn this into a Booking aggregate tailor made to deal with the expected business logic. Business logic will reasonably address concerns in the area of permissions, priorities, costs and even plain concurrency. However, at the very minimum the saga method has to create a Booking aggregate and save it.

At first sight, the code snippet in **Figure 3** is no different from a plain CRUD except for its use of a factory and the outstanding Repository property. The combined effect of factory and repository writes in the configured event stores all the events triggered within the implementation of the Booking class.

In the end, the repository doesn't save a record with the current state of a Booking class where properties are in some way mapped to columns. It just saves business events to the store and in the end at this stage you know exactly what's happened to your booking (when it was created and how), but you don't have all the classic information ready to display to the user. You know what's happened, but you don't have anything ready to show. The source code of the factory is shown in **Figure 4**.

No properties of the newly created instance of the Booking class are touched in the factory, but an event class is created and populated with the actual data to store in the instance, including the capitalized name of the customer and the unique ID that will permanently track the reservation throughout the system. The event is passed to the RaiseEvent method, part of the MementoFX framework, because it's the base class of all aggregates. RaiseEvent adds the event to an internal list that the repository will go through when "saving" the instance of the aggregate. I used the term "saving" because that's just what happens, but I put quotes around it to emphasize that it's a different type of action than in a classic CRUD. The repository saves the event that a reservation was created with the specified data. More precisely, the repository saves all the events logged to an instance of the aggregate during the execution of a business workflow, namely a saga handler method, as shown in **Figure 5**.

But tracking the business event resulting from a command is not enough.

Figure 3 **Structure of a Saga Class**

```
public class ReservationSaga : Saga,
  IAmStartedBy<MakeReservationCommand>,
  IHandleMessages<ChangeReservationCommand>,
  IHandleMessages<CancelReservationCommand>
{
   ...
  public void Handle(MakeReservationCommand msg)
  {
    var slots = CalculateActualNumberOfSlots(msg);
    var booking = Booking.Factory.New(
      msg.FullName, msg.When, msg.Hour, msg.Mins, slots);
    Repository.Save(booking);
  }
}
```

Figure 4 **Source Code of the Factory**

```
public static class Factory
{
  public static Booking New(string name, DateTime when,
    int hour, int mins, int length)
  {
    var created = new NewBookingCreatedEvent(
      Guid.NewGuid(), name.Capitalize(), when,
      hour, mins, length);

    // Tell the aggregate to log the "received" event
    var booking = new Booking();
    booking.RaiseEvent(created);
    return booking;
  }
}
```

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

STOP SHARING!

1&1 VIRTUAL SERVER CLOUD

starting at **$4.99** per month*

intel XEON inside

Trusted Performance.
Intel® Xeon® processors.

NEW

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

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

1 CALL SPEAK WITH AN EXPERT 24/7

1 MONTH FLEXIBLE PAYMENT OPTIONS

1 SECURE CHOICE FAIL-SAFE

1-844-296-2059

1&1®

1and1.com

* 1&1 Virtual Server Cloud S: $4.99/month. Billing cycle 1 month. Minimum contract term 12 months. No set up free.

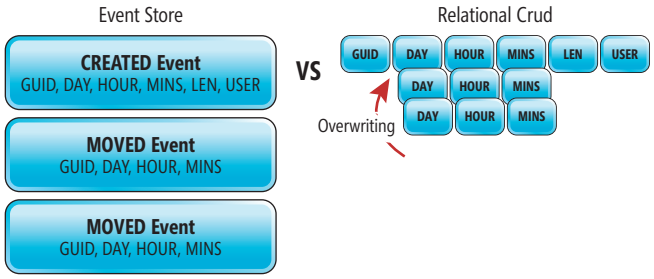


Figure 5 **Saving Events Versus Saving State**

Denormalizing Events to the Query Stack

If you look at CRUD through the lens of preserving data history, you see that creating and reading entities don't affect history, but the same can't be said for updating and deleting. An event store is append-only and updates and deletions are just new events related to the same aggregates. Having a list of events for a given aggregate, though, tells you everything about the history except the current state. And the current state is just what you need to present to users.

Here's where denormalizers fit in. A denormalizer is a class built as a collection of event handlers, just like those being saved to the event store. You register a denormalizer with the bus and the bus dispatches events to it every time it gets one. The net effect is that a denormalizer written to listen to the created event of a booking is given a chance to react whenever one is triggered.

A denormalizer gets the data in the event and does whatever you need it to do, for example, keeping an easy-to-query relational database in sync with recorded events. The relational database (or a NoSQL store or a cache, if that's easier or more beneficial to use) belongs to the query stack and its API isn't given access to the stored list of events. What's more, you can have multiple denormalizers creating ad hoc views of the same raw events. (I'll delve deeper into this aspect in my next column.) In **Figure 1**, the calendar from which a user picks a slot is populated from a plain relational database that's maintained in sync with events by the action of a denormalizer. See **Figure 6** for the denormalizer class code.

Figure 6 **Structure of a Denormalizer Class**

```
public class BookingDenormalizer :
  IHandleMessages<NewBookingCreatedEvent>,
  IHandleMessages<BookingMovedEvent>,
  IHandleMessages<BookingCanceledEvent>
{
  public void Handle(NewBookingCreatedEvent message)
  {
    var item = new BookingSummary()
    {
      DisplayName = message.FullName,
      BookingId = message.BookingId,
      Day = message.When,
      StartHour = message.Hour,
      StartMins = message.Mins,
      NumberOfSlots = message.Length
    };

    using (var context = new MfxbiDatabase())
    {
      context.BookingSummaries.Add(item);
      context.SaveChanges();
    }
  }
  ...
}
```

With reference to **Figure 5**, denormalizers provide a relational CRUD for read purposes only. The output of denormalizers is often called the "read model." Entities in the read model don't typically match the aggregates used to generate events as they are mostly driven by the needs of the UI.

Updates and Deletions

Suppose now the user wants to move a previously booked slot. A command is placed with all the details of the new slot and a saga method takes care of writing a Moved event for the given booking. The saga needs to retrieve the aggregate and needs it in the updated state. If denormalizers just created a relational copy of the aggregate's state (therefore the read model nearly coincides with the domain model), you can get the updated state from there. Otherwise, you create a fresh copy of the aggregate and run all logged events on it. At the end of the replay, the aggregate is in the most updated state. Replaying events isn't a task you have to perform directly. In MementoFX you get an updated aggregate with a line of code within a saga handler:

```
var booking = Repository.GetById<Booking>(message.BookingId);
```

> A denormalizer is a class built as a collection of event handlers just like those being saved to the event store.

Next, you apply to the instance any business logic you need. The business logic generates events and events are persisted through the repository:

```
booking.Move(id, day, hour, mins);
Repository.Save(booking);
```

If you use the Domain Model pattern and follow DDD principles, the Move method contains all the domain logic and events. Otherwise, you run a function with any business logic and raise events to the bus directly. By binding another event handler to the denormalizer, you have a chance to update the read model.

The approach isn't different for canceling a booking. The event of canceling a booking is a business event and must be tracked. This means you might want to have a Boolean property in the aggregate to perform logical deletion. In the read model, though, deletion could be blissfully physical depending on whether your application is going to query the read model for canceled bookings. An interesting side effect is that you can always rebuild the read model by replaying events from the beginning or from a recovery point. All it takes is to create an ad hoc tool that uses the event store API to read events and call denormalizers directly.

Using the Event Store API

Look at the selection of the dropdown list in **Figure 2**. The user wants to stretch the booking as long as possible from the starting time. The business logic in the aggregate must be able to figure that out and to do so it must access the list of bookings in the same day

later than the starting time. That's no big deal in a classic CRUD, but MementoFX lets you query for events, as well:

```
var createdEvents = EventStore.Find<NewBookingCreatedEvent>(e =>
  e.ToDateTime() >= date).ToList();
```

The code snippet returns a list of NewBookingCreated events following the given time. However, there's no guarantee that the booking created is still active and hasn't been moved to another slot. You really need to get the updated state of those aggregates. The algorithm is up to you. For example, you can filter out from the list of Created events the bookings no longer active and then get the ID of the remaining bookings. Finally, you check the actual slot against the one you want to stretch while avoiding overlapping. In the source code of this article, I coded all this logic in a separate (domain) service in the command stack.

Wrapping Up

Using CQRS and event sourcing isn't limited to particular systems with high-end requirements for concurrency, scalability and performance. With infrastructure available that lets you work with aggregates and workflows, any of today's CRUD systems can be rewritten in a way that brings many benefits. Those benefits include:

- Preserving history of data
- A more effective and resilient way of implementing business tasks and changing tasks to reflect business changes with limited effort and risk of regression
- Because events are immutable facts, they are trivial to copy and duplicate and even read models can be regenerated programmatically at will

> An interesting side effect is that you can always rebuild the read model by replaying events from the beginning or from a recovery point.

This means that the ECS pattern (or CQRS/ES as it's sometimes referred) has a tremendous potential for scalability. Even more, the MementoFX framework here is helpful because it simplifies common tasks and offers the aggregate abstraction for easier programming.

MementoFX pushes a DDD-oriented approach, but you can use the ECS pattern with other frameworks and other paradigms such as the functional paradigm. There's one more benefit, and probably the most relevant. I'll tell you about it in my next column. ■

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*

Exploring Visual Studio Mobile Center

Thomas Dohmke

Mobile devices are now at the center of the technology ecosystem. Many people feel they can't live without their phones, tablets, and laptops, and use them to organize information for both their personal and their professional lives. The business world is reacting to this, and every company today is bent on transforming itself into a mobile business. In order to be successful and thrive, companies will have to support not just one mobile app, but a multi-app mobile strategy. This has come to be called the *appification* of the modern business—the proliferation of targeted, highly optimized apps designed for a particular group of users or for specific tasks. In an *appified* world, development teams have to deliver more apps with more value to customers, while the mobile platforms that run these apps are continuously evolving and the number of devices is constantly growing. The biggest apps are installed on more than 1 billion devices, with thousands of device configurations based on criteria such as screen size, language, locale or network provider.

To tackle those challenges, Microsoft offers an end-to-end solution based on three pillars: Visual Studio and Xamarin allow you to create mobile and desktop apps for all major platforms. Microsoft Azure provides powerful and flexible cloud services that make it easy to start or extend existing infrastructure assets and manage data between server and client experiences. Finally, the Mobile DevOps process helps you manage the application lifecycle and allows you to automate integration, testing, delivery and monitoring of your app. The current version of this Mobile DevOps stack consists of multiple integrated, but otherwise disparate, tools such as Visual Studio Team Services (VSTS), Xamarin Test Cloud and HockeyApp. Microsoft has learned a lot from offering these products in the past few years, including how developers can adopt Mobile DevOps practices faster and with less friction.

In this article, I am proud to announce the public preview of our product, combining all features into a single platform: Visual Studio Mobile Center.

This article discusses a preview version of Visual Studio Mobile Center. All information is subject to change.

This article discusses:

- The mobile application lifecycle
- Signing up for Visual Studio Mobile Center
- Adding the Mobile Center SDK
- Integrating with Azure App Service

Technologies discussed:

Azure App Service, Visual Studio Mobile Center

The Mobile Application Lifecycle

DevOps practices can cover the whole development process, but they're not an all-or-nothing commitment. Similar to a set of command-line tools, each practice can be used alone, but you get additional value by combining multiple practices. This is exactly how Mobile Center was designed. It starts with a great onboarding experience for developers, including an API-first design, which allows you to use Mobile Center completely through its REST API or command-line interface (CLI). Each feature solves a specific problem and is usable on its own.

With Build, you can take your source code in any Git repository and create an installable app package automatically with every commit or push. Best of all, you don't need to provision any agents or external machines that run macOS to build your iOS apps. Mobile Center takes care of this and will compile your iOS and Android app right from the source code, with no manual setup on your side.

Once the build process is finished, you want to run your tests on real devices. Test Cloud offers more than 400 unique device configurations to validate your app's behavior. Tests can be written in C# (UITest), Ruby (Calabash), or Java (Appium), and are automatically executed after a build succeeds.

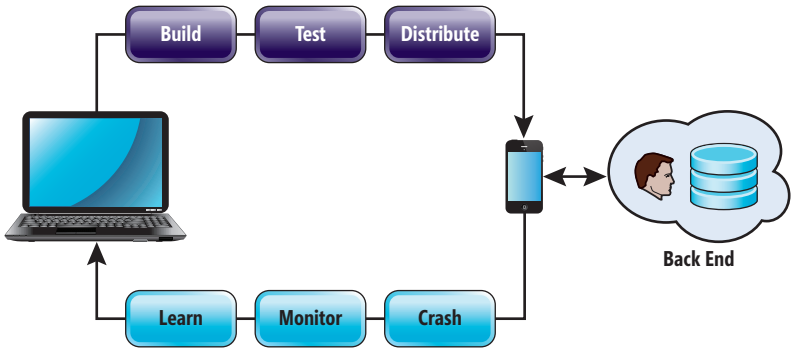


Figure 1 **The Mobile Application Lifecycle with Visual Studio Mobile Center**

When all tests are green, your app needs to get into the hand of testers—which is exactly what the Distribute feature offers. Enter a set of e-mails—called Distribution Groups—and your users can install the app directly on their phones, much as they'd download an app from the app store. Mobile Center Build, Test and Distribute work together seamlessly, so each of your code commits gets the maximum level of validation. And none of these features require any modification of your code.

The next step is to add monitoring to the app. Whether it's automated testing, manual testing, a demo for a customer, or the launch of a new release in the App Store, the collection of usage and diagnostic data is vital for the success of your app. By adding the Mobile Center SDK, you can collect Crashes and leverage Analytics to assess quality issues, learn how and where customers are using your app, and ultimately improve ratings and reviews by responding to their requests.

The SDK also enables two other features: Identity and Tables. Identity is the quickest way to authenticate your app's users with their Google, Facebook, Twitter or Microsoft account. Tables lets you build a cloud store for viewing, adding or modifying app data—even when there's no network connection.

Figure 1 shows how these features enable the mobile application lifecycle.

Getting Started

Signing up for Mobile Center is really easy. If you already have a HockeyApp account, your existing e-mail and password will continue to work and sync between both systems. If not, you could create a new account through the registration form, but instead I want to show you the quickest flow to getting started: Connect with GitHub. If you live on GitHub as I do, you won't need to enter your GitHub password—you'll immediately see the authentication page shown in **Figure 2**. Simply approve access and you'll land on the Mobile Center dashboard with a fully enabled account. No e-mail confirmation, credit card or other setup steps needed.

Once your account is connected to GitHub, you're only a few steps away from your first build. Create your app by simply entering the name, then pick your Git repository and branch via the Build menu as shown in **Figure 3**. Mobile Center will automatically detect your project type, so you don't have to manually add or configure build steps. Behind the scenes, the build pipeline provisions a virtual machine to compile the app, runs all unit tests and signs the app package for distribution.

When the build process completes, you can download the app package file for local installation, execute automated tests on real devices using the Mobile Center Test, or you can continue with the Distribute feature to ship the app to your testers. Each member of your distribution group will receive an e-mail notification to install the app. Simply open the e-mail on the phone and install the app from the browser.

Adding the SDK

For Crashes and Analytics, you need to add the Mobile Center SDK to your project. The SDK is designed with a modular architecture that lets you integrate only those features you want in your app. Let's start with an example for an iOS app. First, add the following import statements to the top of the AppDelegate.swift file:

```
import MobileCenter
import MobileCenterCrashes
import MobileCenterAnalytics
```

The MobileCenterCore is required for all features, while the other two can be omitted if you don't want the Crashes or Analytics features. Next, start the SDK in the didFinishLaunchingWithOptions method:

```
MSMobileCenter.start("{Your App Secret}", withServics:[MSAnalytics.self,
  MSCrashes.self])
```

The App Secret is a GUID that can be taken from the app's management page in Mobile Center. Instead of typing those four lines, you can simply copy them from the app's overview page, as shown in **Figure 4**.

The setup looks very similar on Android:

```
import com.microsoft.azure.mobile.MobileCenter;
import com.microsoft.azure.mobile.crashes.Crashes;
import com.microsoft.azure.mobile.analytics.Analytics;

// ...

MobileCenter.start(getApplication(), "{Your App Secret}", Analytics.class,
  Crashes.class);
```

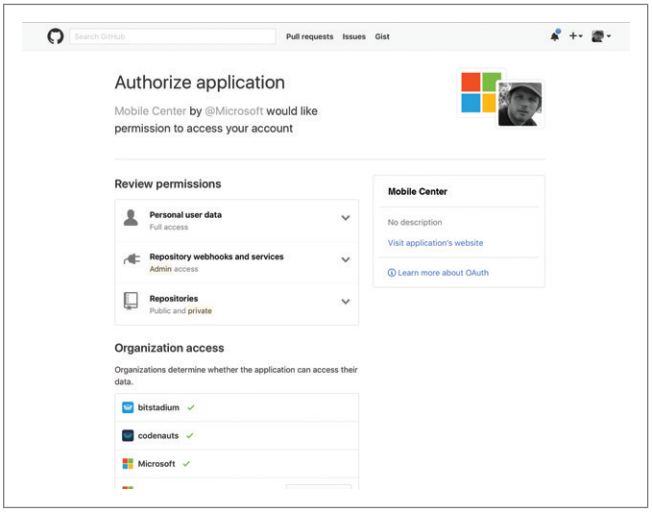


Figure 2 **Signing up for Mobile Center with GitHub**

By enabling the Crashes module, the SDK will automatically start catching unhandled exceptions on iOS and Android, as well as low-level crashes on iOS. To generate a test crash, you can add the following line to a button handler on iOS:

```
MSCrashes.generateTestCrash()
```

The same call is available for Android:

```
Crashes.generateTestCrash();
```

After a crash, the SDK stores the log file in the device's local storage. When the user opens the application again, all pending crash logs will be posted to the Mobile Center back end and processed there. If you build your iOS app with the Mobile Center Build feature, the stack trace in the crash log will be automatically symbolicated, which means all memory addresses are converted into class and method names, file names, and line numbers. If you used a different build server or build your iOS app on your local Mac, you'll see the new crash logs in the Incoming menu first. You can then upload the matching dSYM files to start the symbolication process.

By enabling the Analytics module, the SDK will automatically track sessions, device properties such as OS version, model, or manufacturer, and user properties like language or region, and Mobile Center will show them on the Audience page (see **Figure 5**).

You can track your own custom events and properties with the following line on iOS:

```
MSAnalytics.trackEvent("Video clicked",
  withProperties: ["FileName" : "RickRoll.mp4"])
```

On Android, the properties are specified in a HashMap:

```
Map<String,String>properties=new HashMap<String,String>();
properties.put("FileName", "RickRoll.mp4");

Analytics.trackEvent("Video clicked", properties);
```

All telemetry, crashes, events, and properties are collected anonymously and stored securely in the Azure Cloud, giving you full control over the data your app gathers. To enable an even higher level of trust, the Mobile Center SDK is fully open source and available on github.com/Microsoft for iOS (/MobileCenter-SDK-iOS), Android (/MobileCenter-SDK-Android) and Xamarin (/MobileCenter-SDK-Xamarin). I encourage you to review its source code and file bugs and contribute with pull requests.

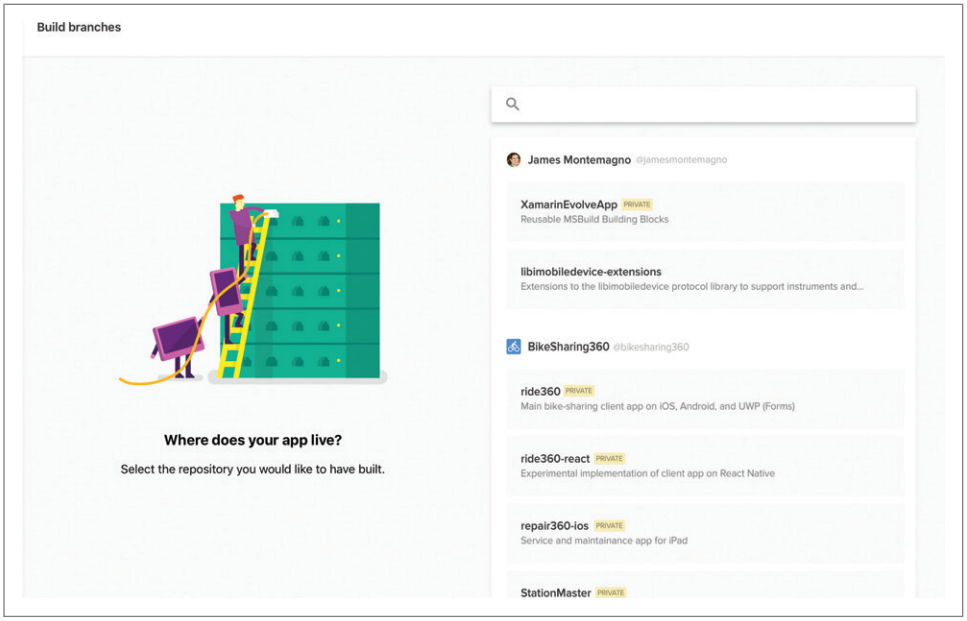


Figure 3 **Selecting a GitHub repository for Build**

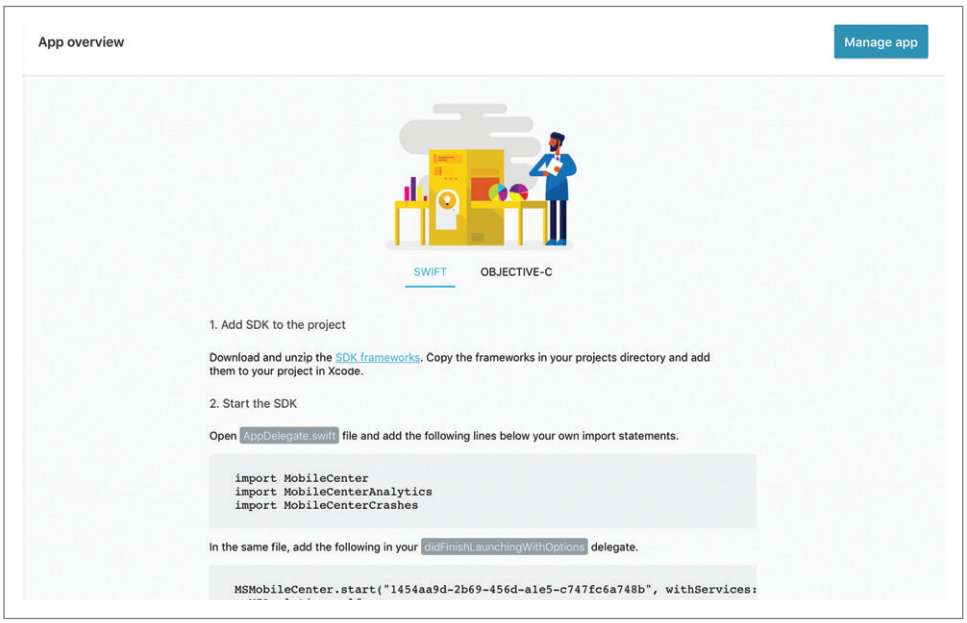


Figure 4 **Getting Started with the Mobile Center SDK**

Back End Included

Most modern mobile apps utilize one or more online services to enable their full power, for example to sync a to-do list among multiple devices, to share data among multiple line-of-business apps, or to restrict features to authenticated users who bought a premium subscription. Mobile Center supports these scenarios by directly integrating with Azure App Service through the Tables and Identity features.

The first time you click on either menu item, you need to connect your Azure subscription. If you don't have one yet, you can sign up for Azure for free on bit.ly/1MoL3Rb. Once your Azure account is linked to Mobile Center, you can create data tables right from the Tables overview page.

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

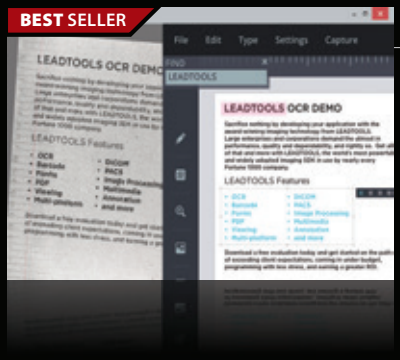
DevExpress DXperience 16.1 | from $1,439.99

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



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

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

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

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

VISA DISCOVER

GSA Schedule
Contract GS-35F-0188R

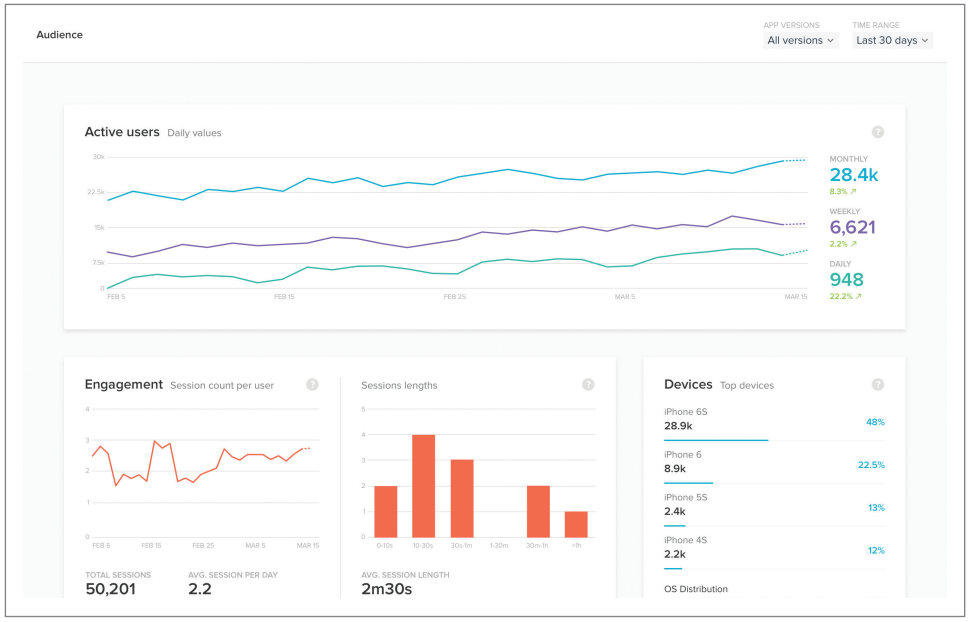


Figure 5 **Mobile Analytics on the Audience Page**

As an example, I create a table with the name "TodoItem" for a to-do app. To access the data in the table from an iOS app, I need to initialize the SDK as follows:

```
let client = MSClient(applicationURLString: "{Your App URL}")
let table = client.tableWithName("TodoItem")
```

As with the App Secret, the App URL should be replaced with the URL for your app from the configuration page in Mobile Center. Then I can query the MSTable object for all available items:

```
table.readWithCompletion { (result, error) in
  if let err = error {
    print("ERROR ", err)
  } else if let items = result?.items {
    for item in items {
      print("Todo Item: ", item["text"])
    }
  }
}
```

To filter the result, I can use an NSPredicate object and the readWithPredicate method, for example, to get only incomplete to-do items:

```
let predicate = NSPredicate(format: "complete == NO")
table.readWithPredicate(predicate) { (result, error) in
  // Handle item
}
```

Inserting a new item is done through an NSDictionary object and the insert method:

```
let newItem = ["text": "Write MSDN article", "complete": false]
table.insert(newItem) { (result, error) in
  if let err = error {
    print("ERROR ", err)
  } else if let item = result {
    print("New item ID: ", item["id"])
  }
}
```

The Azure App Service mobile back end automatically generates new columns based on the dictionary, including a unique id. I can also provide my own id as a key-value pair in the dictionary. Similar to insert, the methods update and delete allow me to modify data, for example, when a to-do item is completed, and delete them. And, of course, all this not only works for iOS apps, but for all supported platforms on Mobile Center.

With the code shown so far, all users of the app would access the same list of to-do items or, more generally, the same data across all devices. This is usually not desirable; typically, each user wants to manage his own to-do list. I can achieve this by adding the Identity feature to my app. Mobile Center supports five identity providers out of the box: Azure Active Directory, Facebook, Google, Microsoft Account and Twitter. An app can use any number of these identity providers to let users sign in with their preferred account. There are two options to integrate the feature:

With a provider SDK (the Facebook SDK, for example), users can sign in to an experience that integrates tightly with the OS on which the app is running. As part of the sign-in process, the app obtains a provider token that needs to be sent back to Azure App Service, which then validates the token and replaces it with a new App Service token. The app can also store the provider token for its own use; to access the Facebook Graph API, for example.

Without a provider SDK, you can use the following method in the Mobile Center SDK:

```
client.loginWithProvider("google", controller: self, animated: true) {
(user, error) in
  // Load data
}
```

This will open a Web view to the provider and the user can sign in. The provider token is directly processed on the server side and the app never receives it. At the end of the flow, the Mobile Center SDK has an App Service token, which is automatically attached to all back-end requests and ensures that the app's data is only accessible by the authenticated user.

Looking Ahead

All features described in this article are available now as a public preview for iOS and Android developers. Sign up for free on aka.ms/mobilecenter. Over the next few months, Microsoft will continuously deliver more features and value, including support for Windows developers, integration with VSTS and more back-end services such as offline data sync. Our goal is that Mobile Center becomes your one-stop shop for the lifecycle of your mobile and desktop apps: build, test, distribute, monitor, and connect to the cloud. My team and I are very excited about the next generation of Mobile DevOps from Microsoft and we hope you'll join us on this journey. ■

Thomas Dohmke *is a co-founder of HockeyApp, which was acquired by Microsoft in late 2014. Dohmke is the group program manager for Visual Studio Mobile Center and is responsible for driving the product vision and managing the team of program managers for each feature. Reach him via e-mail at thdohmke@microsoft.com or on Twittter: @ashtom.*

Scale Your Automated Mobile App Testing with Xamarin Test Cloud

Justin Raczak

In recent years, there's been a dramatic shift in the way teams build and deliver software. Where it was once believed lengthy requirements-gathering processes would ensure the delivery of a perfect product with the first release, it's now known that rapid learning coupled with rapid iteration is the key to success. As the thinking changes, so, too, must the workflows. Development cycles lasting months or years followed by lengthy waterfall QA phases don't facilitate rapid learning. Feedback loops must be shortened and small changes implemented quickly and released to users. To deploy readily, it must be known that software is in a good state at all times. Test automation makes this possible.

Automated testing lets you test your apps in ways that used to take days or weeks. Rather than waiting until the end of a sprint comprised of hundreds of lines of new code, you can test small changes added with every commit. This continual testing surfaces defects as soon as they're introduced and reduces the time needed to debug them. And because the behavior of the app is continually validated, you have the confidence to deploy to users whenever you're ready. Automated testing makes possible a world in which you discover defects and ship a fix within the same day. But the mobile ecosystem presents unique challenges with a diverse landscape of mobile device and OS makers.

This article discusses:

- Preparing the build system and test projects
- Uploading test projects
- Deciding what devices to test on
- Optimizing testing pipelines

Technologies discussed:

Xamarin Test Cloud, Appium

Xamarin Test Cloud makes it fast and simple to scale your automated testing with minimal changes to your existing workflow. Offering more than 400 unique device configurations, Test Cloud enables you to validate your app's behavior on the device models and OS versions that are important to your users without the expense or management overhead that comes with building and managing your own device lab. In most cases, you can tap into this immense value with few to no changes to your code.

Test Cloud supports authoring tests in C# (UITest), Ruby (Calabash) and Java (Appium and Espresso). In the project modification portion of this article, I'll focus on our most requested test framework addition, Appium with JUnit, and walk through the changes you need to make to your project to run your existing tests in Test Cloud. I'll also take a look at the Web interface where you'll review your test results and troubleshoot failed tests. The specific modifications required might change over time. You can find the most recent version of these instructions at bit.ly/2dhp2VQ.

In this example I assume the following preconditions:

- An active Test Cloud account (sign up at bit.ly/2e3YgTy)
- An installed command-line tool (instructions are at bit.ly/2dcrbXS)
- A native Android application project
- An existing suite of Appium tests written in Java with JUnit (at least version 4.9) conforming to Appium 1.5
- Maven build system (at least version 3.3.9)

Changes to the Build System

Before you can begin using Test Cloud, you'll need to add the dependency to ensure the tasks for preparing the requisite files are available to your build.

Add the Test Cloud Dependency To include Test Cloud in your project and ensure its enhanced Android and iOS drivers are available at compile time, add the following dependency to your pom.xml file:

```
<dependency>
  <groupId>com.xamarin.testcloud</groupId>
  <artifactId>appium</artifactId>
  <version>1.0</version>
</dependency>
```

Add the Upload Profile Add the profile from **Figure 1** to your pom.xml inside the <profiles> tag. If you don't already have a <profiles> section, create one and add the profile. This profile will pack your test classes and all dependencies into the target/upload folder where they can then be uploaded to Test Cloud.

Changes to the Tests

Now that your build is configured, you must modify your test classes to leverage the Test Cloud Java extensions.

Add the Imports to Test Classes Import the following packages into your test classes:

```
import com.xamarin.testcloud.appium.Factory;
import com.xamarin.testcloud.appium.EnhancedAndroidDriver;
import org.junit.rules.TestWatcher;
import org.junit.Rule;
```

Instantiate the TestWatcher Insert this instantiation into one of your test classes:

```
@Rule
public TestWatcher watcher = Factory.createWatcher();
```

Update Your Driver Declarations Replace every declaration of AndroidDriver<MobileElement> with EnhancedAndroidDriver<MobileElement>, like so:

```
Private static EnhancedAndroidDriver<MobileElement> driver;
```

Update Your Driver Instantiations Replace every instantiation of your driver such that lines in the form of:

```
Driver = new AndroidDriver<MobileElement>(url, capabilities);
```

become:

```
Driver = new EnhancedAndroidDriver<MobileElement>(url, capabilities);
```

The enhanced driver enables you to "label" the steps in your test using driver.label("myTestStepLabel"). This method will produce a test step label and accompanying screenshot that will be viewable in the test report in Test Cloud. I recommend calling label in the @After method, which will capture a screenshot of the app in its final state before the test completes. The screenshot will be taken even if the test is failing, which might provide valuable insight into why it's failing. In practice, this could look something like:

```
@After
  public void tearDown(){
    driver.label("Stopping app");
    driver.quit();
  }
```

Upload to Test Cloud

Now that your project is equipped with all the prerequisites, you're ready to prepare your files and execute a run in Test Cloud. Before proceeding with the upload steps, it's a good idea to try a local run and make sure everything works as expected. If you need to troubleshoot any of the configuration changes you've just made, it's much faster to do so locally.

To pack your test classes and all dependencies into the target/upload folder, run the following command:

```
mvn -DskipTests -P prepare-for-upload package
```

You might want to verify the target/upload directory now exists in your project's root folder to ensure you're ready for upload. If this will be a new app in Test Cloud, you'll need to create the app as part of the test run. Follow the flow to create a new test run to select your devices, set preferences and generate the command you'll need to execute the run. For this exercise, I recommend selecting a small number of devices from the Tier 1 category so your results will be ready for review quickly. Copy the generated command and run it at the command line.

Figure 1 **Test Cloud Upload Profile**

```
<profile>
  <id>prepare-for-upload</id>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-dependency-plugin</artifactId>
        <version>2.10</version>
        <executions>
          <execution>
            <id>copy-dependencies</id>
            <phase>package</phase>
            <goals>
              <goal>copy-dependencies</goal>
            </goals>
            <configuration>
              <outputDirectory>${project.build.directory}
                /upload/dependency-jars/</outputDirectory>
              <useRepositoryLayout>true</useRepositoryLayout>
              <copyPom>true</copyPom>
            </configuration>
          </execution>
        </executions>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-resources-plugin</artifactId>
        <executions>
          <execution>
            <id>copy-pom-file</id>
            <phase>package</phase>
            <goals>
              <goal>testResources</goal>
            </goals>
            <configuration>
              <outputDirectory>${project.build.directory}
                /upload/</outputDirectory>
              <resources>
                <resource>
                  <directory>
                    ${project.basedir}
                  </directory>
                  <includes>
                    <include>pom.xml</include>
                  </includes>
                </resource>
              </resources>
            </configuration>
          </execution>
          <execution>
            <id>copy-testclasses</id>
            <phase>package</phase>
            <goals>
              <goal>testResources</goal>
            </goals>
            <configuration>
              <outputDirectory>${project.build.directory}
                /upload/test-classes</outputDirectory>
              <resources>
                <resource>
                  <directory>
                    ${project.build.testOutputDirectory}
                  </directory>
                </resource>
              </resources>
            </configuration>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
</profile>
```

Once the file upload has been successfully negotiated and validated, devices will be provisioned, your app will be installed and your tests will execute. The Test Cloud operating model is based on device concurrency, or the number of physical devices that can be used in parallel. For example, a user with five concurrent devices can test an app on the Nexus 5X, Nexus 6P, Samsung Galaxy S5, Samsung Galaxy S6 and HTC M8 at the same time. This efficiency is one of Test Cloud's most significant advantages, making it easy to increase your coverage to span more devices while adding little to no additional wait time.

The command-line tool will stream updates on the test run's status and provide you with a link to the test report once the run has finished. Follow the provided test report link to examine your results.

There are three levels of granularity with which you can view your results:

- Overview report.
- Device grid.
- Device detail report.

I'll discuss each in turn.

The Overview Report The overview provides you with summary information about a test run including pass/fail details, failure stats by OS version, manufacturer, and form factor, and details about the run itself, including device configurations targeted and total run time (see **Figure 2**).

If your test run produces failures, you'll likely want to dig deeper to explore root causes and collect data for debugging. The device grid is the next level of detail.

The Device Grid The device grid provides a highly efficient mechanism for navigating through your test results step by step alongside the screenshots captured at each step. With failures clearly indicated in the test steps, you can quickly jump to a failed step and examine the visual state of your app on each device. For larger device sets, you can filter the devices displayed to only those that failed to create a cleaner field for inspection. If the cause of the failure isn't apparent at this level of detail, you can drill down one more level to view device details (see **Figure 3**).

The Device Detail Report The device detail view gives you the same access to test step navigation and screenshots but provides additional detail specific to the selected device, including CPU and memory usage. From this view you can also access the device logs and stack trace, artifacts that will likely be the most useful when investigating a test failure (see **Figure 4**).

At this point I've followed the most common workflow in Test Cloud:

1. Execute test (manually or via continuous integration, or CI).
2. Review results.
3. Retrieve debugging artifacts.
4. Fix.

Next, I'll discuss a few simple ways to think about your device-targeting strategy and optimizing your testing workflow for performance to keep your pipeline flowing quickly.

Thinking About Device Coverage

Selecting the devices your organization will support and ultimately test against is nearly as important as the testing itself. While there are many sources of aggregate and generalized market data that can help guide

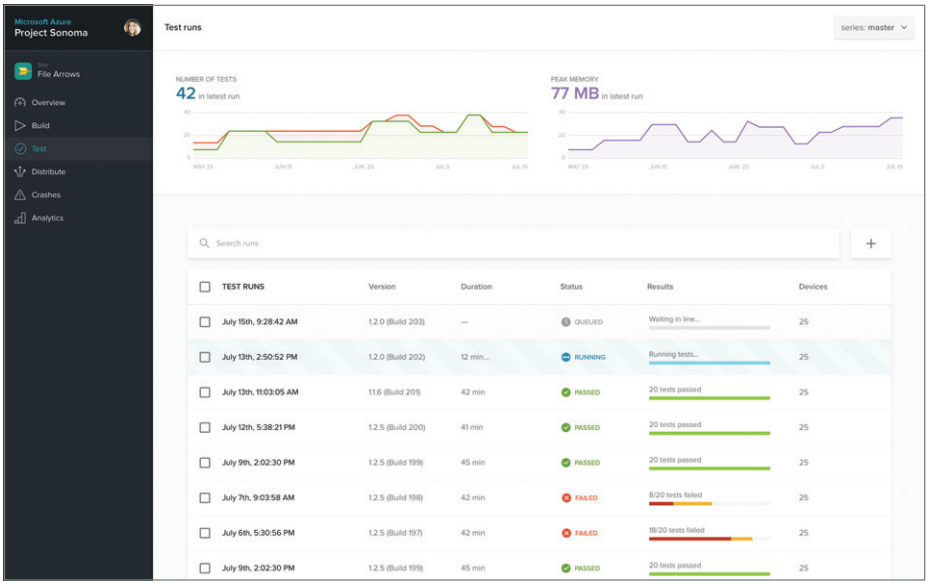


Figure 2 **The Overview Report**

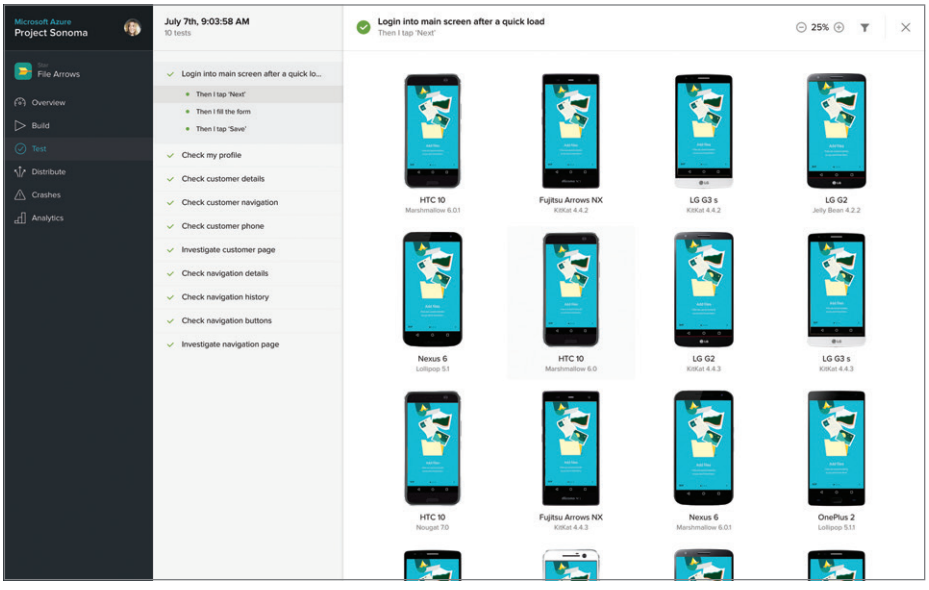


Figure 3 **The Device Grid Report**

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

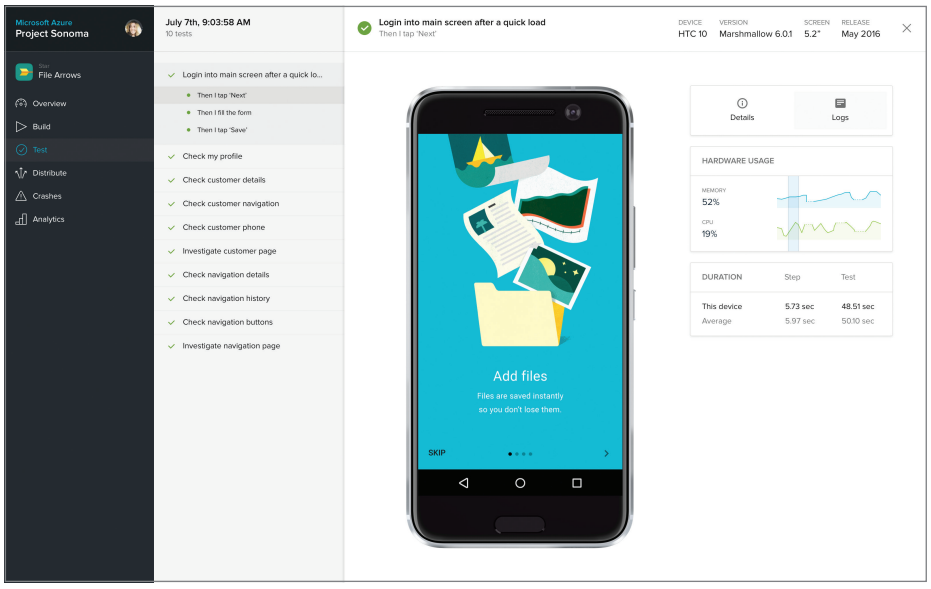


Figure 4 **The Device Detail Report**

your thinking in this area, the most impactful source is usage data from your own user base. The exception to this, of course, is an application only distributed internally to a set of known and managed devices. For external and consumer apps and internal apps distributed under a bring-your-own-device (BYOD) policy, usage data is your best source.

Many tools in the market can help you garner insight into the devices your audience uses. This is the data set from which you can extrapolate your supported device list. The exact methodology you use to determine which devices to support from the aggregate list is up to you and your organization. In most cases, it won't make sense to support every device from your usage data, as this quickly becomes unwieldy and expensive. You might decide to cover as many devices as make up a certain percentage of your user base. Or you might decide to think in terms of numbers of users and support as many devices as required to leave fewer than 500 users' devices covered. If you have an e-commerce app, you might want to cross-reference your usage data with transaction data, ensuring devices that represent the highest spend and most frequent transactions are covered. Again, the specific approach you take to develop your device-support list should be based on the needs and goals of your business.

Keep in mind the mobile market moves quickly. This means, in order for your support list to be accurate and meaningful, you must inspect your usage data regularly. Watch for market signals that might suggest it's a good time to review the data again, such as the rollout of a new device model or OS.

Optimizing Your Testing Pipeline

The best way to extract the most value from test automation is to test early and often. This reduces the time and cost associated with fixing bugs and ensures the deployment pipeline stays clear. But as teams and operations scale, latency can build up in the pipeline and developer productivity can decrease. Let's look at ways to keep the pipeline clear and productivity high.

Not All Tests Are Equal As projects grow over time, their test suites take longer to run. There's an inflection point at which running the test suite after making a simple change becomes painful and cumbersome, often leading to bad habits such as skipping the tests altogether. You can preempt this by thinking early about your application's critical paths—that is, what flows or experiences in your application must absolutely work? Using the earlier e-commerce app example, this might mean users can browse products, add products to the cart and check out. It's less important that users can set their notification preferences. With this structure in place, it becomes much more practical to run tests on every push or even every commit. Instead of running the full test suite for small changes, you can run only those that are part of the critical paths. How, exactly, you accomplish this delineation will depend on the test framework you use.

The Right Devices at the Right Time While testing every push to a feature branch may be ideal from a quality perspective, this quickly becomes expensive for large teams, especially those that support many different device configurations. You can reduce the overhead here by applying a progressive strategy to your device targeting on these test runs. Does a build of a non-production branch need to be tested on every device you support? The answer is likely no. Instead, you can select a sensible number of devices that balances effective testing with shorter wait times. For a pre-production build from CI, a sampling of the most popular device models and OS versions from your device support list will provide a valuable level of coverage without raising your build time beyond an hour. For an individual developer testing from a local workstation, testing against one or two devices might suffice.

These are just a few examples of ways to think about configuring your testing workflow. The broader point is to invest the time to question whether your pipeline flow is optimal. Even if you've answered this question before, as with everything you do, it's always a good idea to routinely inspect and adapt.

Wrapping Up

In this article you've seen how easily you can migrate from running your tests on a simulator or single local device to harnessing the power of hundreds of device configurations using Xamarin Test Cloud. I also touched on a few strategies for organizing your testing workflow and extracting the most value from your test resources. If you're not already using Test Cloud, you can sign up for a free trial at bit.ly/2e3YgTy to begin using it with your projects today. ■

Justin Raczak *is a senior program manager at Microsoft, leading the mobile test automation service. Although he only recently joined Microsoft, he has focused on automated testing and its role in advancing continuous delivery for the past three years. He can be reached at justin.raczak@microsoft.com.*

Thanks to the following Microsoft technical expert for reviewing this article: *Simon Søndergaard*

Visual Studio MAGAZINE

READER'S CHOICE AWARD WINNER 2016

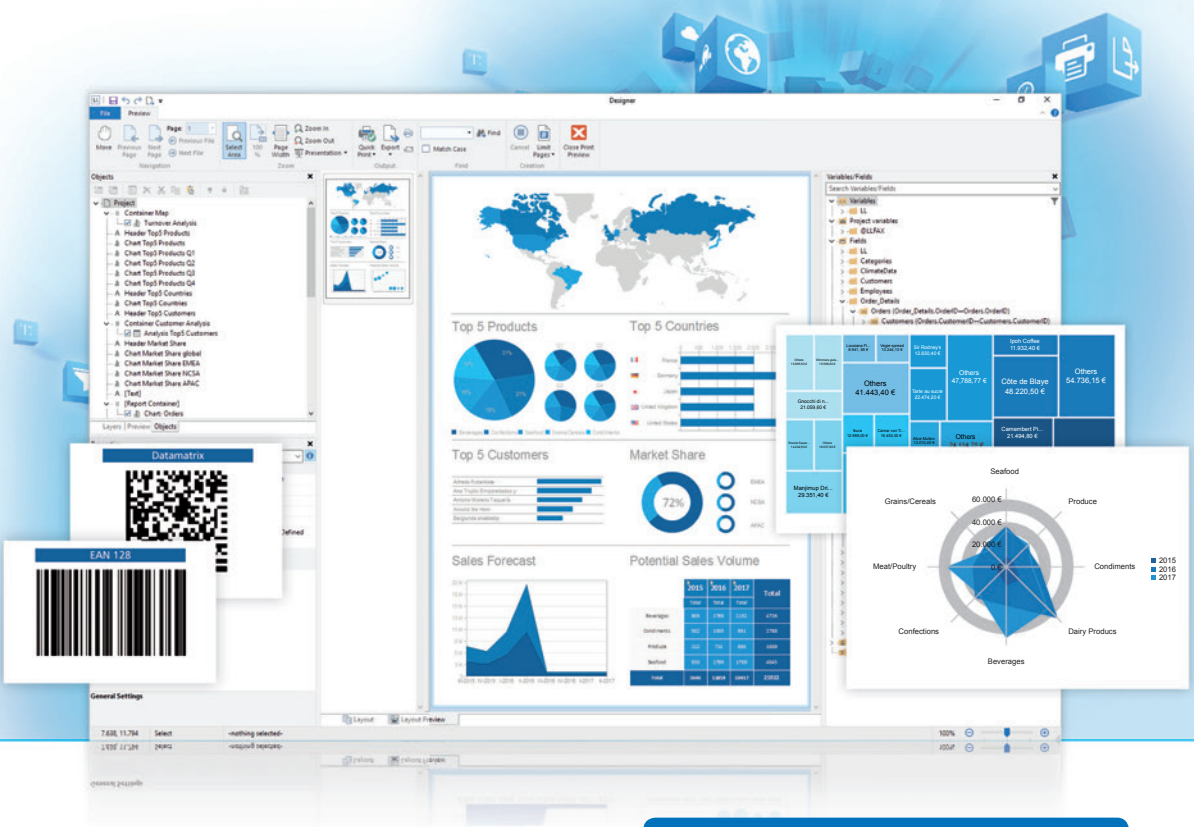
combit®

Visual Studio MAGAZINE 1 READER'S 2016 CHOICE

List & Label Reporting Tool

List & Label enhances applications hassle free, enabling generation of a host of report types. It offers rapid performance and is suitable for deployment in projects large and small. The report designer can be redistributed free of charge.

- Designer can be launched with .NET without a single line of code.
- Supports WinForms, ASP.NET (MVC), WPF.
- Connects to any data source: SQL, ADO.NET, OLE DB, ODBC, ORMs ...
- Designer objects: tables, crosstabs, charts, gauges, barcodes, graphics, PDF, and more.



To learn more, please visit our website → **www.combit.com/reporting**

Extensibility in U-SQL Big Data Applications

Michael Rys

The traditional focus on addressing the big V's of Big Data—volume, velocity and variety—during Big Data processing has mainly concentrated on providing a scalable platform to process the volume of data, on adding near-real-time processing capabilities and on offering the ability to process a variety of input data formats, from CSV over JSON to custom binary formats. One variety that's often been somewhat of an afterthought is the variety associated with custom data processing—not only in terms of format, but also the ability to make it easy to extend your analysis with custom algorithms while preserving the declarative nature of the query language experience.

Some modern Big Data processing and query languages are starting to address this. In particular, U-SQL was designed from the ground up to combine the declarative power of a SQL-based language with the flexibility of using your existing code libraries and developing new custom algorithms.

This article discusses:

- Managing custom code in U-SQL
- Writing a custom reducer
- Processing JSON documents
- Processing image data
- Processing spatial data

Technologies discussed:

U-SQL

In a previous article (bit.ly/10tXM2K), I introduced U-SQL and showed how using the Microsoft .NET Framework type system together with the C#-based expression language in U-SQL makes it seamless to extend your analysis with custom code expressions. I explained how to use C# assemblies to define user-defined functions (UDFs) and use them in your U-SQL query scripts.

U-SQL not only allows you to add your own custom C# functions, but it also provides a framework in which you can add your own user-defined operators (UDOs), such as your own extractors, outputters, and rowset operators, such as processors, appliers, reducers, and custom combiners. The framework consists of two parts:

1. .NET interfaces that provide the contract for you to build these operators in such a way that you can concentrate on your code, leaving the scaled-out execution to U-SQL. Note that the actual business-logic code doesn't have to be implemented in .NET, as I'll show later.
2. U-SQL expressions such as EXTRACT and REDUCE that invoke the custom operators and execute them at scale on your data.

In this article, I'll build on the previous article and show how you can use the U-SQL extensibility mechanisms to process a variety of different data ranging from JSON to image data. I'll also show how to add your own operators.

Managing Your Custom Code in U-SQL

Before I start with some of the examples, let's better understand how U-SQL can use your custom code.

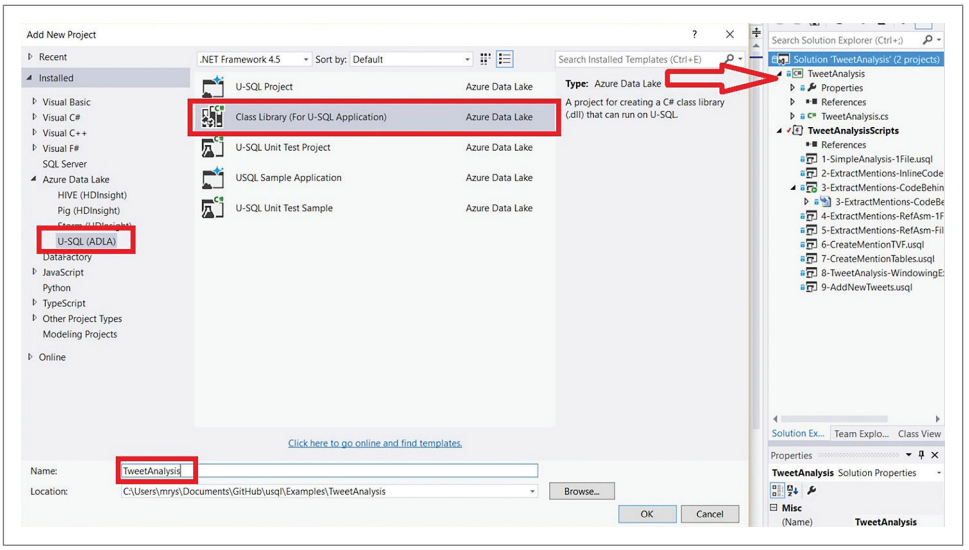


Figure 1 **Class Library (For U-SQL Application) Project**

As mentioned, U-SQL follows C# with its scalar expression language, which is being used in places such as U-SQL predicates and in the expressions in a select clause. For your custom code to become visible to the U-SQL compiler, the code must be packaged into a .NET assembly that must be referenced by the U-SQL script. To be able to reference the assembly, it must have been previously registered in the U-SQL metadata service using a CREATE ASSEMBLY statement.

Registering and Referencing U-SQL Assemblies I suggest using the Azure Data Lake Tools for Visual Studio (aka.ms/adltoolsvs), which make it easy to build and register assemblies that work with U-SQL. If you write your custom code in a "Class Library (For U-SQL Application)" project (see **Figure 1**), you can then write your code and build the project and directly register the generated assembly DLL file with a right-click (see **Figure 2**).

Then all you need in your U-SQL Script is the REFERENCE ASSEMBLY statement to make the public classes and methods usable in your U-SQL script, as shown in **Figure 3**.

Using Existing Code with U-SQL Assemblies Often you'll want to use existing code libraries or even non-.NET code. If you want to use non-.NET code—for example, a native library or even a completely different language runtime like Python or JavaScript—you must wrap the non-.NET code with a C# interoperability layer that will be called from U-SQL and that then calls the non-.NET code, marshaling the data between the components and implementing a UDO interface contract. In this case, the non-.NET code artifacts such as the native .dlls or the files of the different runtime need to be added as additional files. This can be done in the Additional File option of the assembly registration. These files are automatically deployed to every node when the .NET assembly gets referenced in a script and are made available to the .NET assembly's working directory locally to that node.

To use existing .NET libraries, you need to register the existing code libraries as Managed Dependencies on your own assembly, or—if you reuse a library that you can use directly in U-SQL—register it directly in your U-SQL database. In either case, the script must reference all .NET assemblies needed by the script.

I'll show some examples of these registration options in the remainder of the article, as I discuss some custom code scenarios in which it makes sense to use the extensibility model. These scenarios include: merging overlapping ranges with a custom reducer, processing JSON documents, processing image data and processing spatial data. I'll discuss each in turn.

> Often you'll want to use existing code libraries or even non-.NET code.

Merging Overlapping Ranges with a Custom Reducer

Let's assume you have a log file that tracks when a user interacts with your service. Furthermore, let's assume a user can interact with your service in multiple ways (for example, by conducting

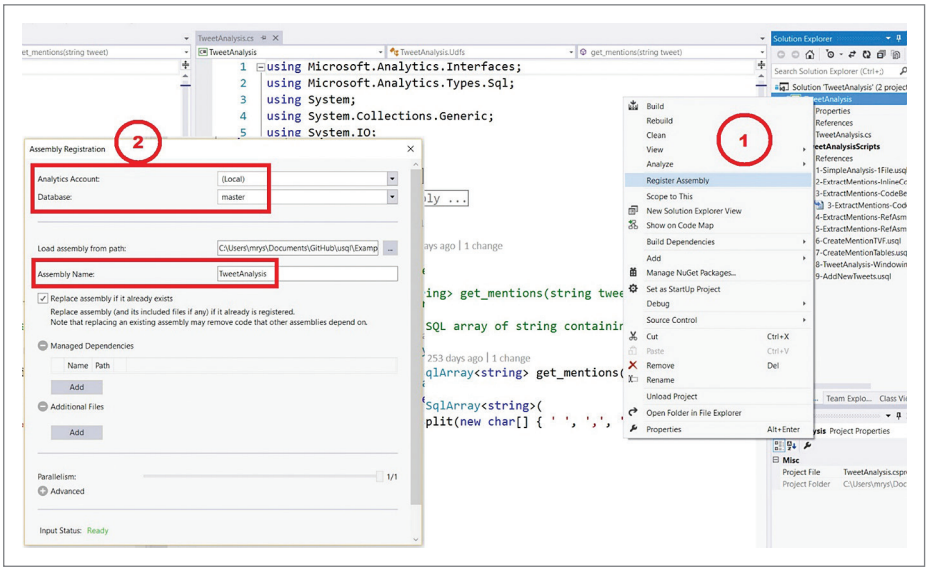


Figure 2 **Registering a U-SQL Assembly**

Figure 3 **Referring to a User-Defined Function from a Custom Assembly**

```
REFERENCE ASSEMBLY master.TweetAnalysis;

USING tweet_fns = TweetAnalysis.Udfs;

@t =
  EXTRACT date string,
          time string,
          author string,
          tweet string
  FROM "/Samples/Data/Tweets/Tweets.csv"
  USING Extractors.Csv();

// Get the mentions from the tweet string
@m =
  SELECT origin
       , tweet_fns.get_mentions(tweet) AS mentions
       , author AS mentioned_by
FROM @t;

...
```

Bing searches from multiple devices or browser windows). As part of your U-SQL job that prepares the log file for later analysis, you want to merge overlapping ranges.

For example, if the input log file looks like **Figure 4**, then you want to merge the overlapping ranges for each user into **Figure 5**.

If you look at the problem, you'll first notice you want to define something like a user-defined aggregation to combine the overlapping time intervals. However, if you look at the input data, you'll notice that because the data isn't ordered, you'll either have to maintain the state for all possible intervals and then merge disjoint intervals as bridging intervals appear, or you need to preorder the intervals for each user name to make the merging of the intervals easier.

The ordered aggregation is simpler to scale out, but U-SQL doesn't provide ordered user-defined aggregators (UDAGGs). In addition, UDAGGs normally produce one row per group, while in this case, I can have multiple rows per group if the ranges are disjoint ranges.

Figure 4 **Log File with Overlapping Time Ranges**

Start Time	End Time	User Name
5:00 AM	6:00 AM	ABC
5:00 AM	6:00 AM	XYZ
8:00 AM	9:00 AM	ABC
8:00 AM	10:00 AM	ABC
10:00 AM	2:00 PM	ABC
7:00 AM	11:00 AM	ABC
9:00 AM	11:00 AM	ABC
11:00 AM	11:30 AM	ABC
11:40 PM	11:59 PM	FOO
11:50 PM	0:40 AM	FOO

Figure 5 **Log File After Merging Overlapping Time Ranges**

Start Time	End Time	User Name
5:00 AM	6:00 AM	ABC
5:00 AM	6:00 AM	XYZ
7:00 AM	2:00 PM	ABC
11:40 PM	0:40 AM	FOO

Luckily, U-SQL provides a scalable UDO called a reducer (bit.ly/2evGsDA) that can aggregate a set of rows based on a grouping key set using custom code.

Let's first write the U-SQL logic where ReduceSample.RangeReducer is our user-defined reducer (Reducer UDO) from the RangeReducer assembly, and the log data is located in the file /Samples/Blogs/MRys/Ranges/ranges.txt (bit.ly/2eseZyw) and uses "-" as the column delimiter. Here's the code:

```
REFERENCE ASSEMBLY RangeReducer;
@in = EXTRACT start DateTime, end DateTime, user string
FROM "/Samples/Blogs/MRys/Ranges/ranges.txt"
USING Extractors.Text(delimiter:'-');

@r =  REDUCE @in PRESORT start ON user
      PRODUCE start DateTime, end DateTime, user string
      READONLY user
      USING new ReduceSample.RangeReducer();

OUTPUT @r
TO "/temp/result.csv"
USING Outputters.Csv();
```

Luckily, U-SQL provides a scalable UDO called a reducer that can aggregate a set of rows based on a grouping key set using custom code.

The REDUCE expression takes the rowset @in as input, partitions it based on the user column, presorts the partitions based on the values in the start column and applies the RangeReducer, producing the same rowset schema on the output. Because the reducer only adjusts the range from start to end, it actually doesn't touch the user column, so you mark it as READONLY. This gives the reducer framework the permission to pass the data through automatically for that column and, in return, allows the U-SQL query processor to aggressively apply optimizations around read-only columns, such as to push predicates on a read-only column ahead of the reducer.

The way to write a reducer is to implement an instance of Microsoft.Analytics.Interfaces.IReducer. In this case, because you don't need to provide any parameters, you only need to overwrite the abstract Reduce method. You can copy the code into a C# Library for U-SQL and register it as the assembly RangeReducer as explained earlier. **Figure 6** shows the implementation of the RangeReducer. (Note that normal code indentation practices have been altered in some code samples because of space constraints.)

The U-SQL REDUCE expression will apply the Reduce method once for each distinct partition key in parallel. The input parameter thus will only contain the rows for a given group and the implementation can return zero to many rows as output.

Because the PRESORT clause guarantees the rows are ordered, the inner logic can assume the data is ordered, and because the user column is marked as READONLY, the column will be passed through automatically and you can write your UDO code more generically by focusing just on the columns you want to transform.

If you now apply the reducer on a large set of data, and if some of your users might be using your system much more frequently than others, you'll encounter something called data skew where some users have large partitions and others only small partitions. Because the contract of the reducer is guaranteed to see all data for that partition, all data must be shuffled to that node and read in one call. Because of this requirement, such data skew in the best case can lead to some partitions taking much longer than others to process, and in the worst case can lead to some reducers running out of the available memory and time resources (a U-SQL vertex will time out after running for about five hours).

If the reducer semantics are associative and commutative and its output schema is the same as its input schema, then a reducer can be marked as recursive, which allows the query engine to split large groups into smaller sub-groups and recursively apply the reducer on these sub-groups to calculate the final result. This recursive application allows the reducer to better balance and parallelize in the presence of data skew. A reducer is marked as recursive by using the property annotation SqlUserDefinedReducer(IsRecursive = true):

```
namespace ReduceSample
{
  [SqlUserDefinedReducer(IsRecursive = true)]
  public class RangeReducer : IReducer
  {
    public override IEnumerable<IRow> Reduce(
      IRowset input, IUpdatableRow output)
    {
      // Insert the code from Figure 6 here
    } // Reduce
  } // RangeReducer
} // ReduceSample
```

In our case, the reducer can be marked as recursive to improve scalability and performance, assuming the processing will preserve the sort among the rows in each recursive invocation.

You can find a Visual Studio project for the example on our GitHub repository at bit.ly/2ecLe5B.

Figure 6 **C# Implementation of the RangeReducer**

```
using Microsoft.Analytics.Interfaces;
using Microsoft.Analytics.Types.Sql;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ReduceSample
{
  public class RangeReducer : IReducer
  {
    public override IEnumerable<IRow> Reduce(
      IRowset input, IUpdatableRow output)
    {
      // Init aggregation values
      bool first_row_processed = false;
      var begin = DateTime.MaxValue;
      var end = DateTime.MinValue;

      // Requires that the reducer is PRESORTED on begin and
      // READONLY on the reduce key.
      foreach (var row in input.Rows)
      {
        // Initialize the first interval with the first row if i is 0
       if (!first_row_processed)
        {
         first_row_processed = true; // Mark that the first row was handled
          begin = row.Get<DateTime>("start");
          end = row.Get<DateTime>("end");
          // If the end is just a time and not a date, it can be earlier
          // than the begin, indicating it is on the next day;
          // this let's you fix up the end to the next day in that case
          if (end < begin) { end = end.AddDays(1); }
        }
        else // Handle the remaining rows
        {
          var b = row.Get<DateTime>("start");
          var e = row.Get<DateTime>("end");
          // Fix up the date if end is earlier than begin
          if (e < b) { e = e.AddDays(1); }

          // If begin time is still inside the interval,
          // increase the interval if it is longer
          if (b <= end)
          {
            // If the new end time is later than the current,
            // extend the interval
            if (e > end) { end = e; }
          }
          else // Output the previous interval and start a new one
          {
            output.Set<DateTime>("start", begin);
            output.Set<DateTime>("end", end);
            yield return output.AsReadOnly();
            begin = b; end = e;
          } // if
        } // if
      } // foreach

      // Now output the last interval
      output.Set<DateTime>("start", begin);
      output.Set<DateTime>("end", end);
      yield return output.AsReadOnly();
    } // Reduce
  } // RangeReducer
} // ReduceSample
```

Processing JSON Documents

One of the most frequent data formats after comma-separated text files is JSON. Unlike CSV file formats, U-SQL doesn't provide a built-in JSON extractor. However, the U-SQL community has provided a sample assembly at bit.ly/2d904va that offers support for extracting and processing both JSON and XML documents.

This solution uses Newtonsoft's Json.NET library (bit.ly/2evWJbz) for the heavy JSON lifting and System.XML for the XML processing. The assembly can extract data from a JSON document using the JsonExtractor (bit.ly/2dPARsM), take a JSON document and split it into a SqlMap to allow navigation and decomposition of JSON documents with the JsonTuple function (bit.ly/2e8tSuX) and finally transform a rowset into a JSON-formatted file with the JSONOutputter (bit.ly/2e4uv3W).

Notice the assembly is designed to be a generic JSON processor, which means it doesn't make any assumption about the JSON document structure and needs to be resilient to the semi-structured nature of JSON, including heterogeneously typed elements (scalar versus structured, different datatypes for the same element, missing elements and so on). If you know your JSON documents adhere to a specific schema, you can possibly create a more efficient JSON extractor.

Unlike in the reducer example earlier, where you write your own assembly that you then deploy, in this case the solution is ready to be used. You can either load the solution from our GitHub repository into Visual Studio and build and deploy it yourself, or you can find the DLLs in the solution's bin\Debug directory.

As mentioned earlier, the non-system dependency requires that both the Samples.Format and the Json.NET assemblies must be registered in the U-SQL metadata store (you can select the Newtonsoft assembly as Managed Dependency when registering the Format

assembly using the Visual Studio tool) and both need to be referenced if you want to process JSON documents. Assuming you've installed your JSON assemblies in your U-SQL catalog under the name [Microsoft.Analytics.Samples.Formats] and [NewtonSoft.Json] in the U-SQL database JSONBlog (see **Figure 7**), you can use the assemblies by referencing them at the beginning of your scripts with:

```
REFERENCE ASSEMBLY JSONBlog.[NewtonSoft.Json];
REFERENCE ASSEMBLY JSONBlog.[Microsoft.Analytics.Samples.Formats];
```

The U-SQL community has provided a sample assembly that offers support for extracting and processing both JSON and XML documents.

The JSON extractor implements the U-SQL IExtractor interface. Because JSON documents need to be fully parsed to make sure they're well-formed, a file containing a single JSON document will need to be processed in a single Extractor vertex. Thus, you indicate that the extractor needs to see the full file content by setting the AtomicFileProcessing property to true (see **Figure 8**). The extractor can be called with an optional parameter called rowpath that allows us to identify the JSON objects that will each be mapped to a row using a JSONPath expression (bit.ly/1EmvgKO).

The extractor implementation will pass the input stream that the U-SQL Extractor framework feeds into the extractor to the Json.NET JsonTextReader. Then it will use the rowpath to get the sub-trees being mapped to a row using SelectChildren. Because JSON objects can be heterogeneous, the code is returning the generic JObject instead of positional JArray or scalar values.

Note that this this extractor is loading the JSON document into memory. If your document is too big, it could cause an out-of-memory condition. In that case, you'd have to write your own extractor that streams through the document without having to load the full document into memory.

Now let's use the JSON Extractor and the JSON tuple function to parse the complex JSON document from /Samples/Blogs/MRys/JSON/complex.json (bit.ly/2ekwOEQ) provided in **Figure 9**.

The format is an array of person "objects" (technically objects with a single person key each) that in turn contain some person properties and address objects. The U-SQL script in **Figure 10** extracts a row per person/address combination.

Notice the script passes the JSONPath expression [*].person to the extractor, thus generating a row for each person field in the top-level array. The EXTRACT schema is being used by the extractor to get the resulting object's properties into columns. Because the addresses field is itself a nested JSON document, the first invocation of the JsonTuple function creates a map containing the address objects, which then are mapped to one row per address with the CROSS APPLY EXPLODE expression. Finally, all the address properties are projected out from the map data type to give you the rowset, as shown in **Figure 11**.

You can find a Visual Studio project of the example and other JSON processing scenarios, including multiple JSON documents inside a file, on our GitHub repository at bit.ly/2dzceLv.

Processing Image Data

In this example, I'm processing some larger unstructured data: images. In particular, I want to process JPEG pictures and extract some of the JPEG EXIF properties, as well as create a thumbnail of the image. Fortunately, .NET provides a variety of image processing capabilities in the System.Drawing class. So all I need to do is build the U-SQL extension functions and operators, delegating the JPEG processing to these classes.

There are several ways to do this. An initial attempt might load all images as byte arrays into a rowset and then apply individual

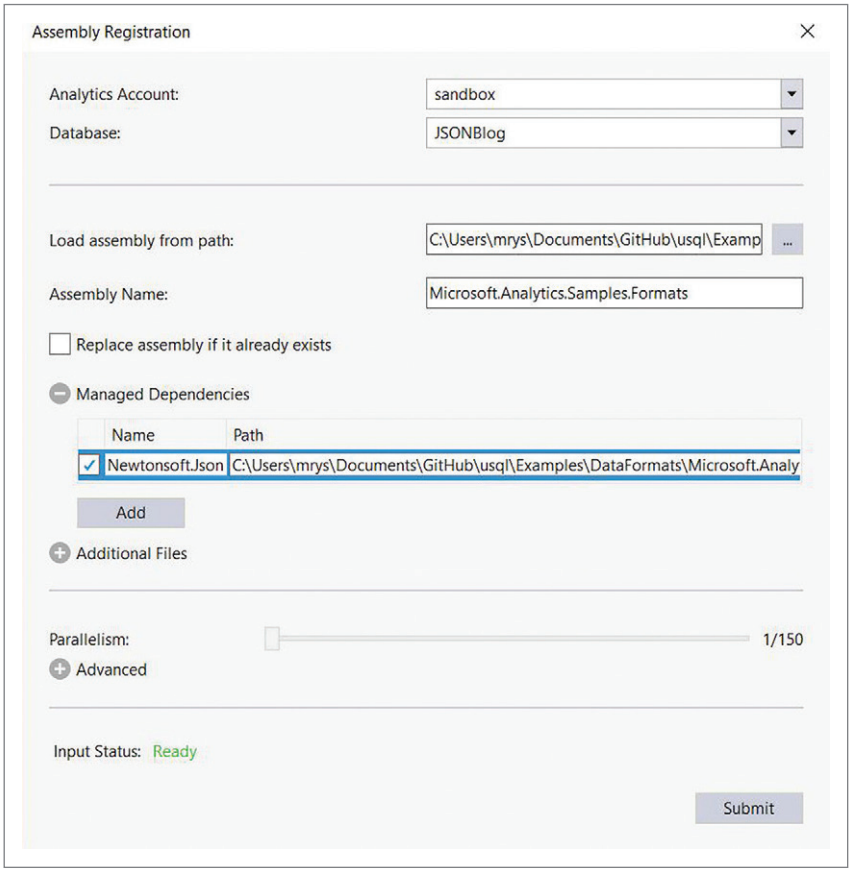


Figure 7 **Registering the Formats Assembly in Visual Studio**

user-defined functions to extract each of the properties and create the thumbnail, as shown in **Figure 12**.

However, this approach has some drawbacks:

- U-SQL rows can be at most 4MB in size, thus limiting the solution to 4MB-sized images (minus the size of the other columns).
- Each of the function invocations can add to the memory pressure and requires flowing the byte array through the U-SQL processing.

Therefore, a better approach is to do the property extraction and thumbnail creation directly inside the custom extractor. **Figure 13** shows a revised U-SQL script.

This script extracts the properties and the thumbnail from the images specified by the file-set pattern (bit.ly/2ektTY6): /Samples/Data/Images/{name}.{format}. The SELECT statement then restricts the extraction to JPEG files by using a predicate only on the format column that will eliminate all non-JPEG files from the extraction (the optimizer will only apply the extractor to the files that satisfy the predicate on the format column). The extractor provides the option to specify the thumbnail's dimensions. The script then outputs the features into a CSV file and uses a simple byte-stream-level outputter to create a thumbnail file for one of the scaled-down images.

Figure 14 shows the implementation of the extractor.

The extractor again needs to see the entire file and operates on the input.BaseStream but now creates only one Image in memory, unlike the script in **Figure 12**. The extractor also checks for each of the requested columns and only processes the data for the requested column names using the extension method SetColumnIfExists.

For more details, see the Visual Studio project on our GitHub site at bit.ly/2dngXCE.

Figure 8 **The JSON Extractor**

```
[SqlUserDefinedExtractor(AtomicFileProcessing = true)]
public class JsonExtractor : IExtractor
{
  private string rowpath;

  public JsonExtractor(string rowpath = null)
  {
    this.rowpath = rowpath;
  }

  public override IEnumerable<IRow> Extract(
    IUnstructuredReader input, IUpdatableRow output)
  {
    // Json.NET
    using (var reader = new JsonTextReader(
      new StreamReader(input.BaseStream)))
    {
      // Parse Json
      var root = JToken.ReadFrom(reader);

      // Rows
      // All objects are represented as rows
      foreach (JObject o in SelectChildren(root, this.rowpath))
      {
        // All fields are represented as columns
        this.JObjectToRow(o, output);

        yield return output.AsReadOnly();
      }
    }
  }
}
```

Figure 9 **A JSON Example Document**

```
[{
  "person": {
    "personid": 123456,
    "name": "Person 1",
    "addresses": {
      "address": [{
        "addressid": "2",
        "street": "Street 2",
        "postcode": "1234 AB",
        "city": "City 1"
      }, {
        "addressid": "2",
        "street": "Street 2",
        "postcode": "5678 CD",
        "city": "City 2"
      }]
    }
  }
}, {
     "person": {
     "personid": 798,
     "name": "Person 2",
     "addresses": {
       "address": [{
         "addressid": "1",
         "street": "Street 1",
         "postcode": "1234 AB",
         "city": "City 1"
     }, {
         "addressid": "4",
         "street": "Street 7",
         "postcode": "98799",
         "city": "City 3"
      }]
   }
  }
}]
```

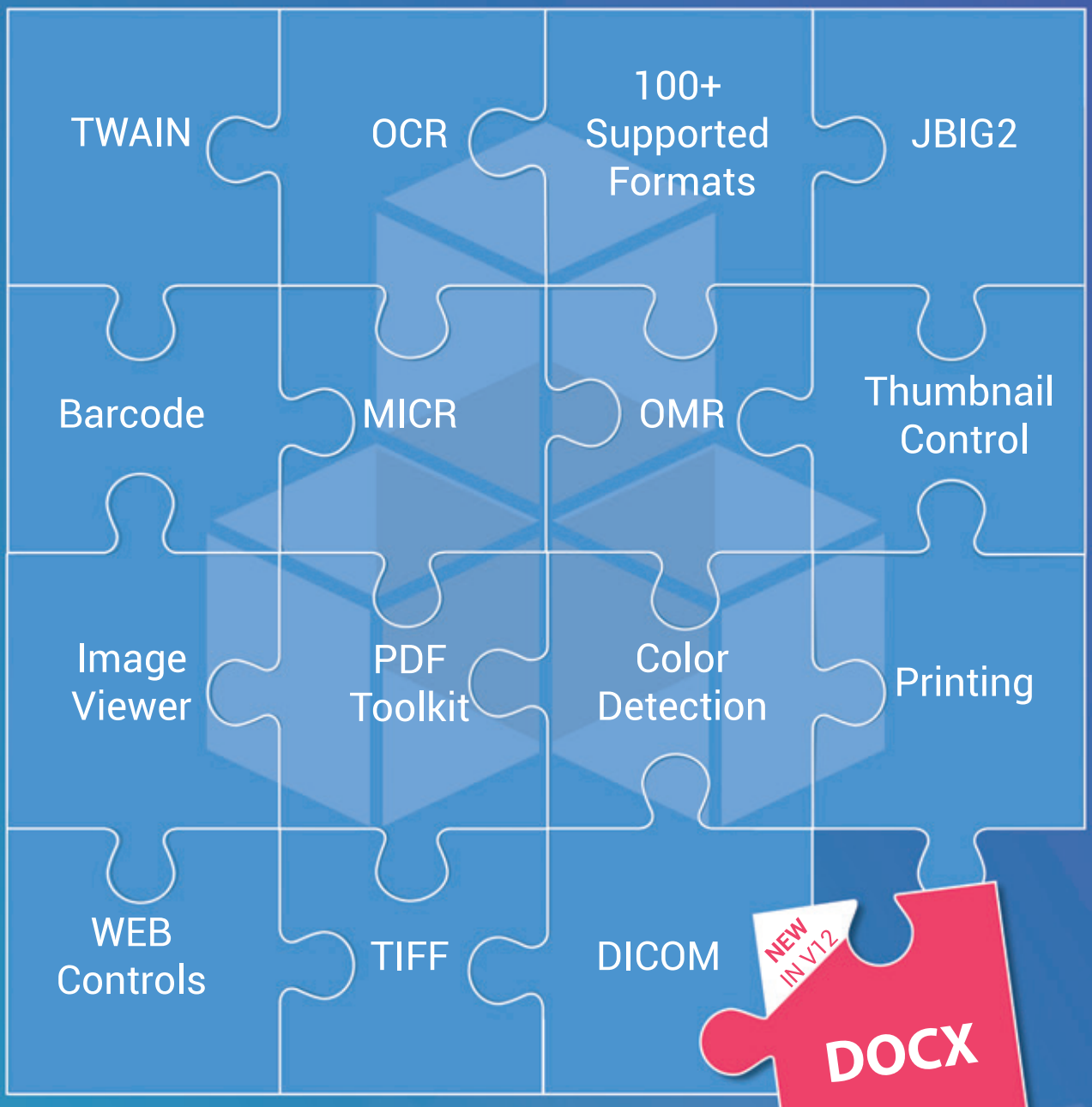
Figure 10 **U-SQL Script Processing the Example JSON Document from Figure 9**

```
DECLARE @input string = "/Samples/Blogs/MRys/JSON/complex.json";

REFERENCE ASSEMBLY JSONBlog.[Newtonsoft.Json];
REFERENCE ASSEMBLY JSONBlog.[Microsoft.Analytics.Samples.Formats];

USING Microsoft.Analytics.Samples.Formats.Json;

@json =
  EXTRACT personid int,
          name string,
          addresses string
  FROM @input
  USING new JsonExtractor("[*].person");

@person =
  SELECT personid,
         name,
         JsonFunctions.JsonTuple(
           addresses, "address")["address"] AS address_array
  FROM @json;

@addresses =
  SELECT personid,
         name,
         JsonFunctions.JsonTuple(address) AS address
  FROM @person
       CROSS APPLY
         EXPLODE (JsonFunctions.JsonTuple(address_array).Values)
           AS A(address);

@result =
  SELECT personid,
         name,
         address["addressid"]AS addressid,
         address["street"]AS street,
         address["postcode"]AS postcode,
         address["city"]AS city
  FROM @addresses;

OUTPUT @result
TO "/output/json/persons.csv"
USING Outputters.Csv();
```

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

Processing Spatial Data

In this example, I'm going to show how to use the SQL Server Spatial type assembly Microsoft.SqlServer.Types.dll in U-SQL. In particular, I want to use the spatial library functions in the U-SQL scripts as user-defined functions. As in the case of the JSON extractor discussed earlier, this means you want to register an already existing assembly in U-SQL without having to write your own assembly.

First, you need to download and install the assembly from the SQL Server 2016 feature pack (bit.ly/2dZTw1k). Select the 64-bit version of the installer (ENU\x64\SQLSysClrTypes.msi) to ensure you have the 64-bit version of the libraries.

The installer installs the managed assembly Microsoft.SqlServer.Types.dll into C:\Program Files (x86)\Microsoft SQL Server\130\SDK\Assemblies and the native assembly SqlServerSpatial130.dll into \Windows\System32\. Next, upload the assemblies into your Azure Data Lake Store (for example, into a folder called /upload/asm/spatial). Because the installer has installed the native library into the system folder c:\Windows\System32, you have to make sure that you either copy SqlServerSpatial130.dll out from that folder before uploading it, or make sure that the tool you use doesn't perform File System Redirection (bit.ly/1TYm9YZ) on system folders. For example, if you want to upload it with the current Visual Studio ADL File Explorer, you'll have to copy the file into another directory first, otherwise—as of the time of the writing of this article—you'll get the 32-bit version uploaded (because Visual Studio is a 32-bit application that does File System Redirection in its ADL upload file selection window), and when you run a U-SQL script that calls into the native assembly, you'll get the following (inner) error at runtime: "Inner exception from user expression: An attempt was made to load a program with an incorrect format. (Exception from HRESULT: 0x8007000B)."

After uploading the two assembly files, register them in a database named SQLSpatial with this script:

```
DECLARE @ASSEMBLY_PATH string = "/upload/asm/spatial/";
DECLARE @SPATIAL_ASM string = @ASSEMBLY_PATH+"Microsoft.SqlServer.Types.dll";
DECLARE @SPATIAL_NATIVEDLL string = @ASSEMBLY_PATH+"SqlServerSpatial130.dll";

CREATE DATABASE IF NOT EXISTS SQLSpatial;
USE DATABASE SQLSpatial;

DROP ASSEMBLY IF EXISTS SqlSpatial;
CREATE ASSEMBLY SqlSpatial
FROM @SPATIAL_ASM
WITH ADDITIONAL_FILES =
  (
    @SPATIAL_NATIVEDLL
  );
```

Note in this case you only register one U-SQL assembly and include the native assembly as a strong dependency to the U-SQL assembly. In order to use the spatial assemblies, you need only reference the U-SQL assembly and the additional file will automatically be made available for the assembly. **Figure 15** shows a simple sample script using the spatial assembly.

Figure 11 **The Rowset Generated by Processing the JSON Document from Figure 9**

123456	Person 1	2	Street 2	1234 AB	City 1
123456	Person 1	2	Street 2	5678 CD	City 2
798	Person 2	1	Street 1	1234 AB	City 1
798	Person 2	4	Street 7	98799	City 3

The SQL Types library has a dependency on the System.Xml assembly, so you need to reference it. Also, some of the methods are using the System.Data.SqlTypes types instead of the built-in C# types. Because System.Data is already included by default, you can simply reference the needed SQL type. The code in **Figure 15** is available on our GitHub site at bit.ly/2dMSBm9.

Wrapping Up: Some Tips and Best Practices for UDOs

This article, while only scratching the surface of the powerful extensibility capabilities of U-SQL, has shown how the U-SQL extensibility mechanism allows you to reuse existing domain-specific code while using the U-SQL extension framework to scale the processing out over the typical Big Data volume.

Figure 12 **Processing Images in U-SQL by Loading Images into Rows**

```
REFERENCE ASSEMBLY Images;

USING Images;

@image_data =
  EXTRACT image_data byte[]  // Max size of row is 4MB!
        , name string
        , format string
  FROM @"/Samples/Data/Images/{name}.{format}"
  USING new ImageExtractor();

// Use UDFs
@image_properties =
  SELECT ImageOps.getImageProperty(image_data, ImageProperties.copyright)
         AS image_copyright,
         ImageOps.getImageProperty(image_data, ImageProperties.equipment_make)
         AS image_equipment_make,
         ImageOps.getImageProperty(image_data, ImageProperties.equipment_model)
         AS image_equipment_model,
         ImageOps.getImageProperty(image_data, ImageProperties.description)
         AS image_description
  FROM @image_data
  WHERE format IN ("JPEG", "jpeg", "jpg", "JPG");
```

Figure 13 **Processing Images in U-SQL by Extracting the Features with an Extractor**

```
REFERENCE ASSEMBLY Images;

@image_features =
  EXTRACT copyright string,
          equipment_make string,
          equipment_model string,
          description string,
          thumbnail byte[],
          name string,
          format string
  FROM @"/Samples/Data/Images/{name}.{format}"
  USING new Images.ImageFeatureExtractor(scaleWidth:500, scaleHeight:300);

@image_features =
  SELECT *
  FROM @image_features
  WHERE format IN ("JPEG", "jpeg", "jpg", "JPG");

OUTPUT @image_features
TO @"/output/images/image_features.csv"
USING Outputters.Csv();

@scaled_image =
  SELECT thumbnail
  FROM @image_features
  WHERE name == "GT4";

OUTPUT @scaled_image
TO "/output/images/GT4_thumbnail_2.jpg"
USING new Images.ImageOutputter();
```

Visual Studio MAGAZINE

READER'S CHOICE AWARD WINNER 2016

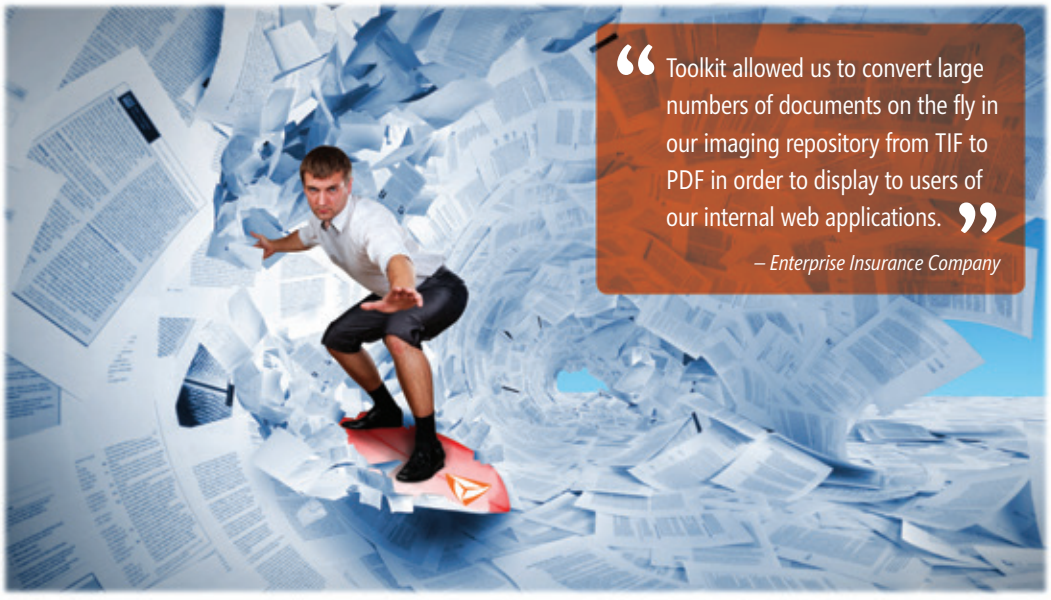
ActivePDF®
DRIVING THE DIGITAL BUSINESS

Visual Studio MAGAZINE 2 READER'S 2016 CHOICE

11 year Award Winner
by Visual Studio Magazine
Reader's Choice Award

ActivePDF Toolkit™

Ride Through the Wave of Paper Documents



ActivePDF Toolkit delivers high-speed, programmatic PDF manipulation through a straightforward .NET and COM interface. For over 16 years, businesses have turned to ActivePDF Toolkit to move structured and unstructured data into high fidelity, searchable, and retrievable documents.

DEVELOPER EFFICIENCY

- ✔ Quick & easy integration into enterprise applications
- ✔ Access coding examples in a variety of programming languages
- ✔ Superior support from evaluation to integration & beyond
- ✔ Saves coding time & stays within budget

CUSTOMIZABLE FEATURES

- ✔ Powerful PDF generator to extract, insert, bookmark, barcode & much more
- ✔ Control PDF fields to include read-only, color, font, & digital signatures
- ✔ XMP to classify & index to meet archiving requirements
- ✔ Merge, encrypt & compress PDF files

Download your FREE 30-day trial → **www.ActivePDF.com**

Toll Free US: 866 468 6733 | Outside US: +1 949 582 9002

Figure 14 **The Image Feature Extractor**

```
using Microsoft.Analytics.Interfaces;
using Microsoft.Analytics.Types.Sql;
using System;
using System.Collections.Generic;
using System.Linq;

using System.IO;
using System.Drawing;
using System.Drawing.Imaging;
using System.Drawing.Drawing2D;

namespace Images
{
  public static class UpdatableRowExtensions
  {
    public static void SetColumnIfExists<T>(this IUpdatableRow source
                                          , string colName, T value)
    {
      var colIdx = source.Schema.IndexOf(colName);
      if (colIdx != -1)
      { source.Set<T>(colIdx, value); }
    }
  }

  [SqlUserDefinedExtractor(AtomicFileProcessing = true)]
  public class ImageFeatureExtractor : IExtractor
  {
    private int _scaleWidth, _scaleHeight;

    public ImageFeatureExtractor(int scaleWidth = 150, int scaleHeight = 150)
    { _scaleWidth = scaleWidth; _scaleHeight = scaleHeight; }

    public override IEnumerable<IRow> Extract(IUnstructuredReader input
                                            , IUpdatableRow output)
    {
      byte[] img = ImageOps.GetByteArrayforImage(input.BaseStream);

      using (StreamImage inImage = new StreamImage(img))
      {
        output.SetColumnIfExists("image", img);
        output.SetColumnIfExists("equipment_make",
          inImage.getStreamImageProperty(ImageProperties.equipment_make));
        output.SetColumnIfExists("equipment_model",
          inImage.getStreamImageProperty(ImageProperties.equipment_model));
        output.SetColumnIfExists("description",
          inImage.getStreamImageProperty(ImageProperties.description));
        output.SetColumnIfExists("copyright",
          inImage.getStreamImageProperty(ImageProperties.copyright));
        output.SetColumnIfExists("thumbnail",
          inImage.scaleStreamImageTo(this._scaleWidth, this._scaleHeight));
      }
      yield return output.AsReadOnly();
    }
  }
}
```

But such a powerful tool can also be misused easily, so here are some tips and best practice advice.

While custom data formats often need a custom extractor and potentially an outputter, one should consider very carefully if the data format can be extracted in parallel (such as CSV-type formats) or if the processing needs to see all the data in a single operator instance. Additionally, making the operators generic enough so processing only happens if a specific column is requested can also potentially improve performance.

When considering UDOs such as processors, reducers, combiners and appliers, it's highly recommended to first consider a pure U-SQL solution that leverages built-in operators. For example, the range reducer script discussed earlier could actually be written with some clever use of windowing and ranking functions. Here are some reasons why you still might want to consider UDOs:

- The logic needs to dynamically access the input or output schema of the rowset that's being processed. For example, create a JSON document for the data in the row where the columns aren't known ahead of time.
- A solution using several user-defined functions in the SELECT expression creates too much memory pressure and you can write your code to be more memory-efficient in a processor UDO.
- You need an ordered aggregator or an aggregator that produces more than one row per group, and you can't write either with windowing functions.

Figure 15 **Using the Spatial Capabilities in U-SQL**

```
REFERENCE SYSTEM ASSEMBLY [System.Xml];
REFERENCE ASSEMBLY SQLSpatial.SqlSpatial;

USING Geometry = Microsoft.SqlServer.Types.SqlGeometry;
USING Geography = Microsoft.SqlServer.Types.SqlGeography;
USING SqlChars = System.Data.SqlTypes.SqlChars;

@spatial =
    SELECT * FROM (VALUES
                  // The following expression is not using the native DDL
                  ( Geometry.Point(1.0,1.0,0).ToString()),
                  // The following expression is using the native DDL
                  ( Geometry.STGeomFromText(
                    new SqlChars("LINESTRING (100 100, 20 180, 180 180)"),
                    0).ToString())
                 ) AS T(geom);

OUTPUT @spatial
TO "/output/spatial.csv"
USING Outputters.Csv();
```

When considering UDOs such as processors, reducers, combiners and appliers, it's highly recommended to first consider a pure U-SQL solution that leverages built-in operators.

When you're using UDOs, you should always keep the following tips in mind:

- Use the READONLY clause to allow pushing predicates through UDOs.
- Use the REQUIRED clause to allow column pruning to be pushed through UDOs.
- Hint Cardinality on your query expression that uses a UDO, should the query optimizer choose the wrong plan. ■

Michael Rys *is a principal program manager at Microsoft. He has been doing data processing and query languages since the 1980s. He has represented Microsoft on the XQuery and SQL design committees and has taken SQL Server beyond relational with XML, Geospatial and Semantic Search. Currently he's working on Big Data query languages such as SCOPE and U-SQL when he's not enjoying time with his family underwater or at autocross. Follow him on Twitter: @MikeDoesBigData.*

Thanks to the following Microsoft technical experts for reviewing this article: *Michael Kadaner, Saveen Reddy, Clemens Szyperski and Ed Triou*

Visual Studio MAGAZINE

READER'S CHOICE AWARD WINNER 2016

SPARX SYSTEMS

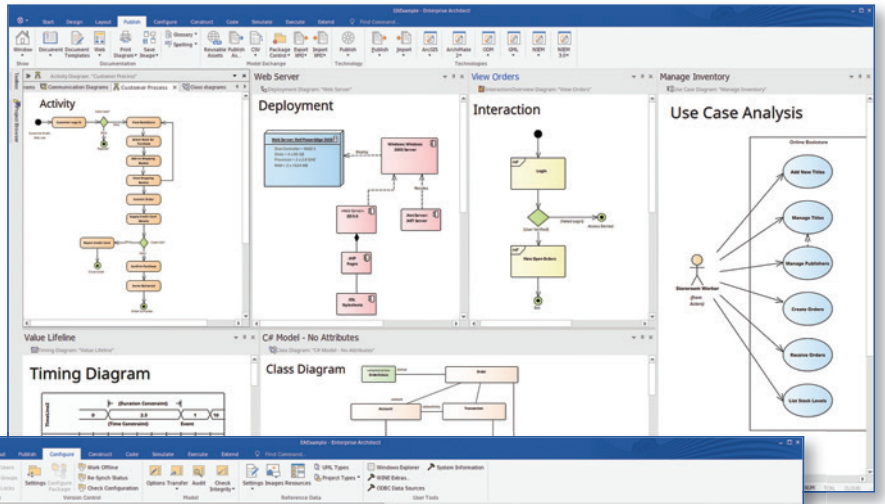
Visual Studio MAGAZINE 2 READER'S 2016 CHOICE

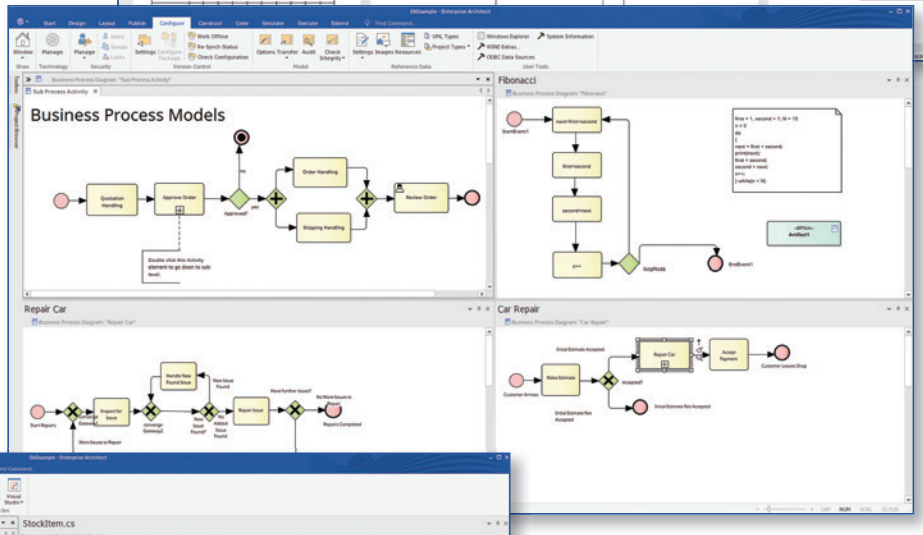
Model and Design Your Creative Vision

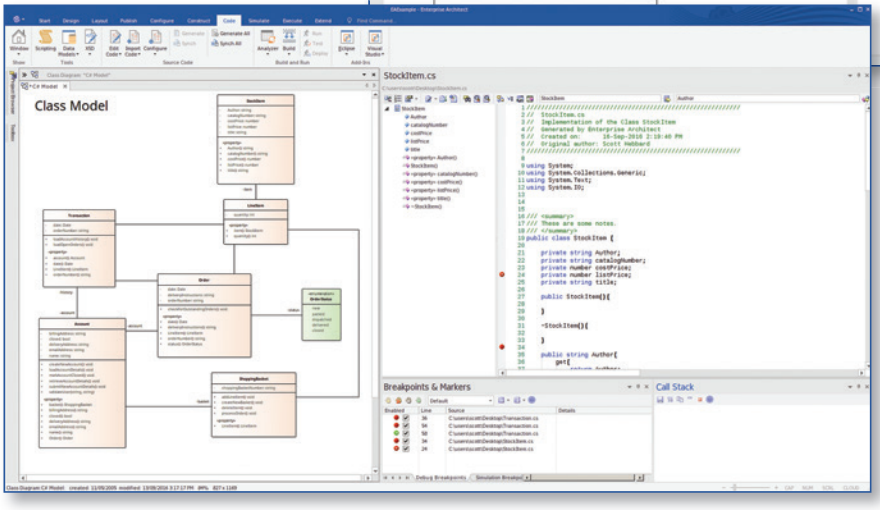
Make your vision a reality by capturing it in Enterprise Architect—your **perfect enterprise wide solution** to visualize, analyze, model, test and maintain all of your systems, software, processes and architectures.

Enterprise Architect is **the ideal platform** to help you to stay in control of your workspace, support your colleagues and team, enable collaboration and build confidence within your most complex projects.

From requirements to implementation and beyond, Sparx Systems' Enterprise Architect is a **fully featured tool suite** that lets you model, design, simulate, prototype, build, test, manage and trace from vision to solution.







ENTERPRISE ARCHITECT

To learn more, visit
www.sparxsystems.com

General JavaScript with Roslyn and T4 Templates

Nick Harrison

The other day my daughter told me a joke about a conversation between a smartphone and a feature phone. It goes something like this: What did the smartphone say to the feature phone? "I'm from the future, can you understand me?" Sometimes it feels like that when learning something new and cutting edge. Roslyn is from the future and can be difficult to understand at first.

In this article, I'll discuss Roslyn in a way that might not get as much focus as it deserves. I'll focus on using Roslyn as a source of metadata for generating JavaScript with T4. This will use the Workspace API, some of the Syntax API, the Symbol API and a runtime template from the T4 engine. The actual JavaScript generated is secondary to understanding the processes used to gather the metadata.

Because Roslyn also provides some nice options for generating code, you might think that the two technologies would clash and not work well together. Technologies often clash when their sandboxes overlap, but these two technologies can play together rather nicely.

This article discusses:

- How to retrieve semantic details about a codebase
- Using a T4 template to map out how you want a JavaScript class to look like
- Using the MSBuildWorkspace to add the generated code to the project

Technologies discussed:

Roslyn, SemanticModel, T4, Workspace API

Wait, What's T4?

If T4 is new to you, a 2015 e-book from the Syncfusion Succinctly series, "T4 Succinctly," provides all the background you need (bit.ly/2cOtWuN).

For now, the main thing to know is that T4 is Microsoft's template-based text transformation toolkit. You feed metadata to the template and the text becomes the code you want. Actually, you aren't limited to code. You can generate any type of text, but source code is the most common output. You can generate HTML, SQL, text documentation, Visual Basic .NET, C# or any text-based output.

Look at **Figure 1**. It shows a simple Console Application program. In Visual Studio, I added a new runtime text template named AngularResourceService.tt. The template code automatically generates some C# code that'll implement the template at run time, which you can see in the console window.

In this article, I'll show you how to use Roslyn to gather metadata from a Web API project to feed to T4 to generate a JavaScript class and then use Roslyn to add that JavaScript back to the solution.

Conceptually, the process flow will look like **Figure 2**.

Roslyn Feeds T4

Generating code is a metadata-hungry process. You need metadata to describe the code you want generated. Reflection, Code Model and the Data Dictionary are common sources of readily available

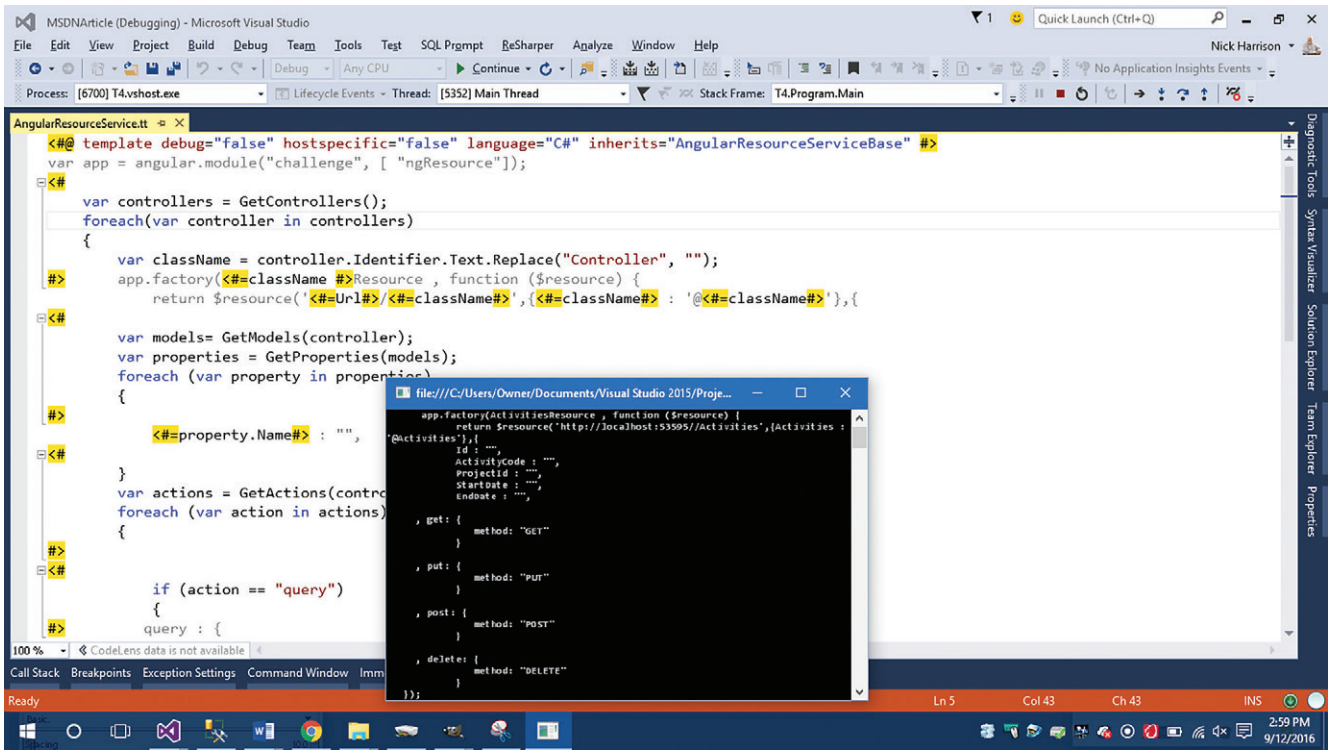


Figure 1 **Using T4 for Design-Time Code Generation**

metadata. Roslyn can provide all the metadata you would've received from Reflection or the Code Model but without some of the problems these other approaches incur.

In this article, I'll use Roslyn to find classes derived from ApiController. I'll use the T4 template to create a JavaScript class for each Controller and expose a method for each Action and a property for each property in the ViewModel associated with the Controller. The result will look like the code in **Figure 3**.

Gathering the Metadata

I start gathering metadata by creating a new console application project in Visual Studio 2015. In this project, I'll have a class devoted to gathering metadata with Roslyn, as well as a T4 template. This will be a runtime template that will generate some JavaScript code based on the gathered metadata.

Once the project is created, the following commands from the Package Manager Console are issued:

```
Install-Package Microsoft.CodeAnalysis.CSharp.Workspaces
```

This ensures the latest Roslyn code for the CSharp compiler and related services are being used.

I place the code for the various methods in a new class called RoslynDataProvider. I'll refer to this class throughout the article and it'll be a handy reference whenever I want to gather metadata with Roslyn.

I use the MSBuildWorskspace to get a workspace that will provide all the context needed for the compilation. Once I have the solution, I can easily walk through the projects looking for the WebApi project:

```
private Project GetWebApiProject()
{
  var work = MSBuildWorkspace.Create();
  var solution = work.OpenSolutionAsync(PathToSolution).Result;
  var project = solution.Projects.FirstOrDefault(p =>
    p.Name.ToUpper().EndsWith("WEBAPI"));
  if (project == null)
    throw new ApplicationException(
      "WebApi project not found in solution " + PathToSolution);
  return project;
}
```

If you follow a different naming convention, you can easily incorporate it into the GetWebApiProject to find the project in which you're interested.

Now that I know which project I want to work with, I need to get the compilation for that project, as well as a reference to the type that

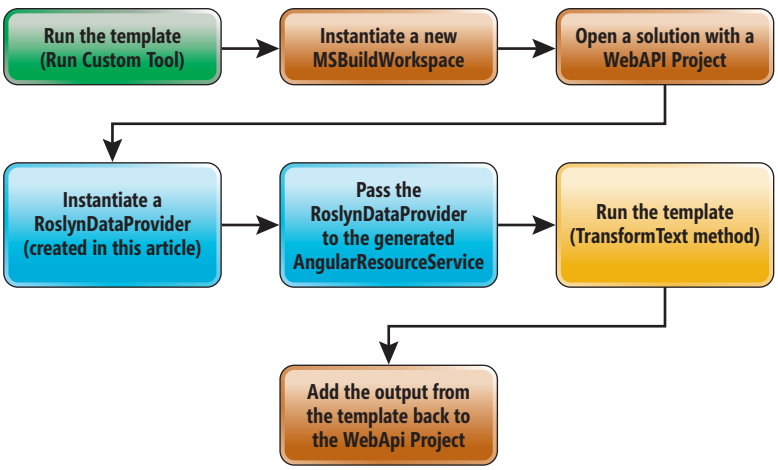


Figure 2 **T4 Process Flow**

Figure 3 **Result of Running the Code**

```
var app = angular.module("challenge", [ "ngResource"]);
  app.factory(ActivitiesResource , function ($resource) {
    return $resource(
      'http://localhost:53595//Activities',{Activities : '@Activities'},{
    Id : "",
    ActivityCode : "",
    ProjectId : "",
    StartDate : "",
    EndDate : "",

  , get: {
      method: "GET"
    }

  , put: {
      method: "PUT"
    }

  , post: {
      method: "POST"
    }
  , delete: {
      method: "DELETE"
    }
  });
});
```

I'll use to identify the controllers of interest. I need the compilation because I'll use the SemanticModel to determine whether a class derives from System.Web.Http.ApiController. From the project, I can get the documents included in the project. Each document is a separate file, which could include more than one class declaration, although it's a good best practice to only include a single class in any file and have the name of the file match the name of the class; but not everyone follows this standard all the time.

Finding the Controllers

Figure 4 shows how to find all of the class declarations in every document and determine if the class is derived from ApiController.

Because the compilation has access to all of the references needed to compile the project, it will have no problem resolving the target type. When I get the compilation object, I've started compiling the project, but am interrupted part way through once I have the details to get the needed metadata.

Figure 5 shows the GetBaseClasses method that does the heavy lifting for determining if the current class derives from the target class. This does a bit more processing than is strictly needed. To determine whether a class is derived from ApiController, I don't really care about the interfaces implemented along the way, but by including these details, this becomes a handy utility method that can be used in a wide variety of places.

This type of analysis gets complicated with Reflection because a reflective approach will rely on recursion and potentially needing to have to load any number of assemblies along the way to get access to all of the intervening types. This type of analysis isn't even possible with the Code Model, but is relatively straightforward with Roslyn using the SemanticModel. The SemanticModel is a treasure trove of metadata; it represents everything the compiler knows about the code after going through the trouble of binding the syntax trees to symbols. In addition to tracking down base types, it can be used to answer tough questions like overload/override resolution or finding all references to a method or property or any Symbol.

Finding the Associated Model

At this point, I have access to all the Controllers in the project. In the JavaScript class, it's also nice to expose the properties found in the Models returned by the Actions in the Controller. To understand how this works, take a look at the following code, which shows the output from running scaffolding for a WebApi:

```
public class Activity
  {
    public int Id { get; set; }
    public int ActivityCode { get; set; }
    public int ProjectId { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
  }
```

In this case, the scaffolding was run against the Models, as shown in **Figure 6**.

The ResponseType attribute added to the actions will link the ViewModel to the Controller. Using this attribute, you can get the name of the model associated with the action. As long as the Controller was created using scaffolding, then every action will be associated with the same model, but Controllers created by hand or edited after being generated might not be so consistent. **Figure 7** shows how to compare against all actions to get a complete list of the models associated with a Controller in case there's more than one.

There's interesting logic going on in this method; some of it's rather subtle. Remember what the ResponseType attribute looks like:

```
[ResponseType(typeof(Activity))]
```

I want to access the properties in the type referenced in the type of expression, which is the first parameter to the attribute—in this case, Activity. The attributes variable is a list of the ResponseType attributes found in the controller. The parameters variable is a list

Figure 4 **Finding the Controllers in a Project**

```
public IEnumerable<ClassDeclarationSyntax> FindControllers(Project project)
{
  compilation = project.GetCompilationAsync().Result;
  var targetType = compilation.GetTypeByMetadataName(
    "System.Web.Http.ApiController");

  foreach (var document in project.Documents)
  {
    var tree = document.GetSyntaxTreeAsync().Result;
    var semanticModel = compilation.GetSemanticModel(tree);
    foreach (var type in tree.GetRoot().DescendantNodes().
      OfType<ClassDeclarationSyntax>()
      .Where(type => GetBaseClasses(semanticModel, type).Contains(targetType)))
    {
      yield return type;
    }
  }
}
```

Figure 5 **Finding Base Classes and Interfaces**

```
public static IEnumerable<INamedTypeSymbol> GetBaseClasses
  (SemanticModel model, BaseTypeDeclarationSyntax type)
{
  var classSymbol = model.GetDeclaredSymbol(type);
  var returnValue = new List<INamedTypeSymbol>();
  while (classSymbol.BaseType != null)
  {
    returnValue.Add(classSymbol.BaseType);
    if (classSymbol.Interfaces != null)
      returnValue.AddRange(classSymbol.Interfaces);
    classSymbol = classSymbol.BaseType;
  }
  return returnValue;
}
```

Enterprise-Proven Distributed Caching

Trusted for over a decade, the easiest most powerful in-memory data grid to scale your .NET applications

- **Fast and linearly scalable**
- **Enterprise-grade availability**
- **Industry-leading ease of use**
- **Integrated in-memory computing**

Replacing AppFabric Caching?
Try our source-code compatible drop-in.
www.scaleoutsoftware.com/appfabric

Brought to you by the scalability architects

Step up to a battle-tested in-memory data grid that has been hardened by over 425 enterprise customer deployments. ScaleOut's technology makes advanced features such as parallel LINQ query and integrated MapReduce accessible to any .NET developer. Automatic configuration and turnkey global data replication deliver legendary ease-of-use. ScaleOut's world-class support meets the needs of mission-critical applications. Run on premises or in the cloud on Microsoft Azure or Amazon AWS.

ScaleOut Software

Download your free trial today!
www.scaleoutsoftware.com/trial

Please visit scaleoutsoftware.com to learn how ScaleOut's technology can help accelerate your business or call us at +1.503.643.3422

Figure 6 **Generated API Controller**

```
public class ActivitiesController : ApiController
  {
    private ApplicationDbContext db = new ApplicationDbContext();
    // GET: api/Activities
    public IQueryable<Activity> GetActivities()
    {
      return db.Activities;
    }
    // GET: api/Activities/5
    [ResponseType(typeof(Activity))]
    public IHttpActionResult GetActivity(int id)
    {
      Activity activity = db.Activities.Find(id);
      if (activity == null)
      {
        return NotFound();
      }
      return Ok(activity);
    }
    // POST: api/Activities
    [ResponseType(typeof(Activity))]
    public IHttpActionResult PostActivity(Activity activity)
    {
      if (!ModelState.IsValid)
      {
        return BadRequest(ModelState);
      }
      db.Activities.Add(activity);
      db.SaveChanges();
      return CreatedAtRoute("DefaultApi", new { id = activity.Id }, activity);
    }
    // DELETE: api/Activities/5
    [ResponseType(typeof(Activity))]
    public IHttpActionResult DeleteActivity(int id)
    {
      Activity activity = db.Activities.Find(id);
      if (activity == null)
      {
        return NotFound();
      }
      db.Activities.Remove(activity);
      db.SaveChanges();
      return Ok(activity);
    }
```

of the parameters to these attributes. Each of these parameters will be a TypeOfExpressionSyntax, and I can get the associated type through the type property of the TypeOfExpressionSyntax objects. Once again, the SemanticModel is used to pull in the Symbol for that type, which will give all the details you could want.

Distinct at the end of the method will ensure that each model returned is unique. In most circumstances, you'd expect to have duplicates because multiple actions in the Controller will be associated with the same model. It's also a good idea to check against the ResponseType being void. You won't find any interesting properties there.

Examining the Associated Model

The following code shows how to find the properties from all of the models found in the Controller:

```
public IEnumerable<ISymbol> GetProperties(IEnumerable<TypeInfo> models)
{
  return models.Select(typeInfo => typeInfo.Type.GetMembers()
    .Where(m => m.Kind == SymbolKind.Property))
    .SelectMany(properties => properties).Distinct();
}
```

Finding the Actions

In addition to showing the properties from the associated Models, I want to include references to the methods that are in the Controller. The methods in a Controller are Actions. I'm only interested in the Public methods, and because these are WebApi Actions, they should all be translated to the appropriate HTTP Verb.

There are a couple of different conventions followed for handling this mapping. The one followed by the scaffolding is for the method name to start with the verb name. So the put method would be PutActivity, the post method would be PostActivity, the delete method would be DeleteActivity, and there will generally be two get methods: GetActivity and GetActivities. You can tell the difference between the get methods by examining the return types for these methods. If the return type directly or indirectly implements the IEnumerable interface, the get method is the get all; otherwise, it's the get single item method.

The other approach is that you explicitly add attributes to specify the verb, then the method could have any name. **Figure 8** shows the code for GetActions that identifies the public methods and then maps them to verbs using both methods.

The GetActions method first tries to map based on the name of the method. If that doesn't work, it'll then try to map by attributes. If the method cannot be mapped, then it won't be included in the list of actions. If you have a different convention that you want to check against, you can easily incorporate it into the GetActions method. **Figure 9** shows the implementations for the MapByMethodName and MapByAttribute methods.

Both methods start by explicitly searching for the Get Action and determining which type of "get" to which the method refers.

Figure 7 **Finding Models Associated with a Controller**

```
public IEnumerable<TypeInfo> FindAssociatedModel
  (SemanticModel semanticModel, TypeDeclarationSyntax controller)
{
  var returnValue = new List<TypeInfo>();
  var attributes = controller.DescendantNodes().OfType<AttributeSyntax>()
    .Where(a => a.Name.ToString() == "ResponseType");
  var parameters = attributes.Select(a =>
    a.ArgumentList.Arguments.FirstOrDefault());
  var types = parameters.Select(p=>p.Expression).OfType<TypeOfExpressionSyntax>();
  foreach (var t in types)
  {
    var symbol = semanticModel.GetTypeInfo(t.Type);
    if (symbol.Type.SpecialType == SpecialType.System_Void) continue;
    returnValue.Add( symbol);
  }
  return returnValue.Distinct();
}
```

Figure 8 **Finding the Actions on a Controller**

```
public IEnumerable<string> GetActions(ClassDeclarationSyntax controller)
{
  var semanticModel = compilation.GetSemanticModel(controller.SyntaxTree);
  var actions = controller.Members.OfType<MethodDeclarationSyntax>();
  var returnValue = new List<string>();
  foreach (var action in actions.Where
        (a => a.Modifiers.Any(m => m.Kind() == SyntaxKind.PublicKeyword)))
  {
    var mapName = MapByMethodName(semanticModel, action);
    if (mapName != null)
      returnValue.Add(mapName);
    else
    {
      mapName = MapByAttribute(semanticModel, action);
      if (mapName != null)
        returnValue.Add(mapName);
    }
  }
  return returnValue.Distinct();
}
```

If the action isn't one of the "gets," MapByAttribute checks to see if the action has an attribute that starts with Http. If one's found, then the verb can be determined by simply taking the attribute name and removing Http from the attribute name. There's no need to check explicitly against each attribute to determine which verb to use.

MapByMethodName is structured similarly. After first checking for a Get action, this method uses a regular expression to see if any of the other verbs match. If a match is found, you can get the verb name from the named capture group.

Both of the mapping methods need to differentiate between the Get Single and Get All Actions and they both use the IdentifyEnumerable method shown in the following code:

```
private static bool IdentifyIEnumerable(SemanticModel semanticModel,
  MethodDeclarationSyntax actiol2n)
{
  var symbol = semanticModel.GetSymbolInfo(action.ReturnType);
  var typeSymbol = symbol.Symbol as ITypeSymbol;
  if (typeSymbol == null) return false;
  return typeSymbol.AllInterfaces.Any(i => i.Name == "IEnumerable");
}
```

Again, the SemanticModel plays a pivotal role. I can differentiate between the get methods by examining the return type of the method. The SemanticModel will return the symbol bound to the return type. With this symbol, I can tell whether the return type implements the IEnumerable interface. If the method returns a List<T> or an Enumerable<T> or any type of collection, it'll implement the IEnumerable interface.

The T4 Template

Now that all of the metadata is gathered, it's time to visit the T4 template that will tie all these pieces together. I start by adding a Runtime Text Template to the project.

For a Runtime Text Template, the output of running the template will be a class that will implement the template that's defined and not the code I want to produce. For the most part, anything you can do in a Text Template you can do with a Runtime Text Template. The difference is how you run the template to generate code. With a Text Template, Visual Studio will handle running the template and creating the hosting environment in which the template will run. With a Runtime Text Template, you're responsible for setting up the hosting environment and running the template. This puts more work on you, but it also gives you a great deal more control over how you run the template and what you do with the output. It also removes any dependencies on Visual Studio.

I start by editing the AngularResource.tt and adding the code in **Figure 10** to the template.

Depending on how familiar you are with JavaScript, this might be new to you—if so, don't worry.

The first line is the template directive and it tells T4 that I'll be writing template code in C#; the other two attributes are ignored for Runtime templates, but for the sake of clarity, I'm explicitly stating that I have no expectation from the hosting environment and don't expect the intermediate files to be preserved for debugging.

The T4 template is a bit like an ASP page. The <# and #> tags delimit between code to drive the template and text to be transformed by the template. The <#= #> tags delimit a variable substitution that is to be evaluated and inserted into the generated code.

Looking at this template, you can see that the metadata is expected to provide a className, URL, a list of properties and a list of actions. Because this is a Runtime template there are a couple of things that I can do to simplify matters, but first take a look at the code that's created when this template is run, which is done by saving the .TT file or by right-clicking on the file in Solution Explorer and selecting Run Custom Tool.

The output from running the template is a new class, which matches the template. More important, if I scroll down, I'll find that the template also generated the base class. This is important because if I move the base class to a new file and explicitly state the base class in the template directive, it'll no longer be generated and I'm free to change this base class as needed.

Next, I'll change the template directive to this:

```
<#@ template debug="false" hostspecific="false" language="C#"
  inherits="AngularResourceServiceBase" #>
```

Then I'll move the AngularResourceServiveBase to its own file. When I run the template again, I'll see that the generated class still derives from the same base class, but it was no longer generated. Now I'm free to make any changes needed to the base class.

Next, I'll add a few new methods and a couple of properties to the base class to make it easier to provide the metadata to the template.

To accommodate the new methods and properties, I'll also need a few new using statements:

```
using Microsoft.CodeAnalysis;
using Microsoft.CodeAnalysis.CSharp.Syntax;
```

I'll add properties for the URL and for the RoslynDataProvider that I created at the beginning of the article:

```
public string Url { get; set; }
public RoslynDataProvider MetadataProvider { get; set; }
```

Figure 9 **MapByName and MapByAttribute**

```
private static string MapByAttribute(SemanticModel semanticModel,
  MethodDeclarationSyntax action)
{
  var attributes = action.DescendantNodes().OfType<AttributeSyntax>().ToList();
  if ( attributes.Any(a=>a.Name.ToString() == "HttpGet"))
    return IdentifyIEnumerable(semanticModel, action) ? "query" : "get";
  var targetAttribute = attributes.FirstOrDefault(a =>
    a.Name.ToString().StartsWith("Http"));
  return targetAttribute?.Name.ToString().Replace("Http", "").ToLower();
}

private static string MapByMethodName(SemanticModel semanticModel,
  MethodDeclarationSyntax action)
{
  if (action.Identifier.Text.Contains("Get"))
    return IdentifyIEnumerable(semanticModel, action) ? "query" : "get";
  var regex = new Regex("\b(?'verb'post|put|delete)", RegexOptions.IgnoreCase);
  if (regex.IsMatch(action.Identifier.Text))
    return regex.Matches(action.Identifier.Text)[0]
      .Groups["verb"].Value.ToLower();
  return null;
}
```

Figure 10 **Initial Template**

```
<#@ template debug="false" hostspecific="false" language="C#" | #>
var app = angular.module("challenge", [ "ngResource"]);
  app.factory(<#=className #>Resource . function ($resource) {
    return $resource('<#=Url#>/<#=className#>',{<=className#> : '@<#=className#>'},{
    <#=property.Name#> : "",
  query : {
    method: "GET"
    , isArray : true
    }
  ' <#=action#>: {
    method: "<#= action.ToUpper()#>
    }
  });
});
```

Figure 11 **Helper Methods Added to AngularResourceServiceBase**

```
public IList<ClassDeclarationSyntax> GetControllers()
{
  var project = MetadataProvider.GetWebApiProject();
  return MetadataProvider.FindControllers(project).ToList();
}

protected IEnumerable<string> GetActions(ClassDeclarationSyntax controller)
{
  return MetadataProvider.GetActions(controller);
}

protected IEnumerable<TypeInfo> GetModels(ClassDeclarationSyntax controller)
{
  return MetadataProvider.GetModels(controller);
}

protected IEnumerable<ISymbol> GetProperties(IEnumerable<TypeInfo> models)
{
  return MetadataProvider.GetProperties(models);
}
```

Figure 12 **Final Version of the Template**

```
<#@ template debug="false" hostspecific="false" language="C#"
inherits="AngularResourceServiceBase" #>
var app = angular.module("challenge", [ "ngResource"]);
<#
  var controllers = GetControllers();
  foreach(var controller in controllers)
  {
    var className = controller.Identifier.Text.Replace("Controller", "");
#>    app.facctory(<#=className #>Resource , function ($resource) {
      return $resource('<#=Url#>/<#=className#>',{<#=className#> :
'@<#=className#>'},{
<#
    var models= GetModels(controller);
    var properties = GetProperties(models);
    foreach (var property in properties)
    {
#>
      <#=property.Name#> : "",
<#
    }
    var actions = GetActions(controller);
    foreach (var action in actions)
    {
#>
<#
      if (action == "query")
      {
#>      query : {
      method: "GET"
```

Figure 13 **Running a Runtime Text Template**

```
private static void Main()
{
  var work = MSBuildWorkspace.Create();
  var solution = work.OpenSolutionAsync(Path to the Solution File).Result;
  var metadata = new RoslynDataProvider() {Workspace = work};
  var template = new AngularResourceService
  {
    MetadataProvider = metadata,
    Url = @"http://localhost:53595/"
  };
  var results = template.TransformText();
  var project = metadata.GetWebApiProject();
  var folders = new List<string>() { "Scripts" };
  var document = project.AddDocument("factories.js", results, folders)
    .WithSourceCodeKind(SourceCodeKind.Script)
    ;
  if (!work.TryApplyChanges(document.Project.Solution))
    Console.WriteLine("Failed to add the generated code to the project");
  Console.WriteLine(results);

  Console.ReadLine();
}
```

With these pieces in place, I'll also need a couple of methods that will interact with the MetadataProvider, as shown in **Figure 11**.

Now that I have these methods added to the base class, I'm ready to extend the template to use them. Look at how the template changes in **Figure 12**.

Running the Template

Because this is a Runtime template, I'm responsible for setting up the environment for running the template. **Figure 13** shows the code needed to run the template.

The class created when I save the template or run the custom tool can be instantiated directly and I can set or access any public properties or call any public methods from the base class. This is how the values for the properties are set. Calling the TransformText method will run the template and return the generated code as a string. The results variable will hold the generated code. The rest of the code deals with adding a new document to the project with the code that was generated.

There's a problem with this code, however. The call to AddDocuments successfully creates a document and places it in the scripts folder. When I call the TryApplyChanges, it returns successful. The problem comes when I look in the solution: There's a factories file in the scripts folder. The problem is that instead of factories.js, it's factories.cs. The AddDocument method isn't configured to accept an extension. Regardless of the extension, the document will be added based on the type of project to which it's added. This is by design.

Therefore, after the program runs and generates the JavaScript classes, the file will be in the scripts folder. All I have to do is change the extension from .cs to .js.

Wrapping Up

Most of the work done here centered on getting metadata with Roslyn. Regardless of how you plan to use this metadata, these same practices will come in useful. As for the T4 code, it'll continue to be relevant in a variety of places. If you want to generate code for any language not supported by Roslyn, T4 is a great choice and easy to incorporate into your processes. This is good because you can use Roslyn to generate code for only C# and Visual Basic .NET while T4 lets you generate any type of text, which could be SQL, JavaScript, HTML, CSS, or even plain-old text.

It's nice to generate code like these JavaScript classes because they're tedious and error-prone. They also easily conform to a pattern. As much as is feasible, you want to follow that pattern as consistently as possible. Most important, the way you want that generated code to look is likely to change over time, especially for something new, as best practices are formed. If all you have to do is update a T4 template to change to the new "best way to do it," you're more likely to adhere to the emerging best practices; but if you have to modify large amounts of hand-generated monotonous, tedious code, you'll probably have multiple implementation as each iteration of the best practice was in vogue. ■

Nick Harrison *is a software consultant living in Columbia, S.C., with his loving wife Tracy and daughter. He's been developing full stack using .NET to create business solutions since 2002. Contact him on Twitter: @Neh123us, where he also announces his blog posts, published articles and speaking engagements.*

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

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

Alachisoft HYPERGRID SMARTBEAR Syncfusion

GOLD SPONSORS

GOVERLAN IDERA QuickBase

SUPPORTED BY

Visual Studio

LAST CHANCE!

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 WITH DISCOUNT CODE L360DEC AND **SAVE $300!**

Must use discount code L360DEC

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, BI and Analytics on The Microsoft Stack - *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		
4:00 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	12:00 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	**VSM01** Workshop Continues	
5:00 PM	6:00 PM	EXPO Preview	
6:00 PM	7:00 PM	**Live! 360 Keynote:** Digital Transformation - Is Your IT Career on	
START TIME	**END TIME**		
8:00 AM	9:00 AM	**Visual Studio Live! / Modern Apps Live!** - *Tim Sneath, Principal*	
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** Cloud Oriented Programming - *Vishwas Lele*	**VST14** Creating Cordova Apps using Ionic and Angular 2 - *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** Practical Internet of Things for the Microsoft Developer - *Eric D. Boyd*	**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:** Mobile - Choosing a Direction for Your	
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** Continuous Delivery on Azure: A/B Testing, Canary Releases, and Dark Launching - *Marcel de Vries*	**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	12:00 PM	**VSF01** Workshop: Angular 2 Bootcamp - *John Papa*	
12:00 PM	1:00 PM	Lunch	
1:00 PM	5:00 PM	**VSF01** Workshop Continues	

Speakers and sessions subject to change

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*
VSM02 Workshop Continues	**VSM03** Workshop Continues	**MAM01** Workshop Continues	**ADM01** Workshop Continues	**ADM02** Workshop Continues

Track as Businesses Look to Become More Agile? - *David Foote, Co-founder, Chief Analyst and Chief Research Officer, Foote Partners*

Day 1: Tuesday, December 6, 2016

Keynote: Faster, Leaner, More Productive: The Next Generation of Visual Studio
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** Ensuring Quality Deliverables with SmartBear's Visual Studio Integrations - *Rick Almeida*	**MAT03** Manage Distributed Teams with Visual Studio Team Services and Git - *Brian Randell*	**ADT05** JCP, Adopt-a-JSR and You - *Reza Rahman*	**ADT06** Full Stack Java with JSweet, Angular 2, PrimeNG, and JAX-RS - *Kito Mann*
VST15 Busy Developer's Guide to Chrome Development - *Ted Neward*	**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 Getting Started with Aurelia - *Brian Noyes*	**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** Agile Architecture - *Steve Green*	**ADW04** The Essentials of Building Cloud-Based Web Apps with Azure - *Ido Flatow*

Company - *John Papa, JohnPapa.net, LLC*

VSW11 How to Scale .NET Apps with Distributed Caching - *Iqbal Khan*	**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** Securing Client JavaScript Apps - *Brian Noyes*	**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*	**ADH04** Microservices as Chat Bots Are the Future
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: Building Server Side using .NET Core, MVC, Web API, and Azure - *Allen Conway*	**ADH05** The Curious Case for the Immutable Object - *David Hoerster*	**ADH06** Continuous Integration May Have Negative Effects
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 Client Side using TypeScript and Angular2 - *Allen Conway*	**ADH07** Meeting-Free Software Development in Distributed Teams	**ADH08** Mobile DevOps Demystified with Xamarin, VSTS and HockeyApp - *Roy Cornelissen*
VSH19 Debugging the Web with Fiddler - *Ido Flatow*	**VSH20** User Experience Case Studies - Good and Bad - *Billy Hollis*	**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: Applied Agile - *Philip Japikse*
VSF02 Workshop Continues	**MAF01** Workshop Continues	**ADF01** Workshop Continues

File System Monitoring in Universal Windows Platform Apps

Adam Wilson

The file system of a Windows device changes rapidly. Shared libraries, such as the camera roll, are one of the few places where all processes on the device can simultaneously interact with the same data. Building a Universal Windows Platform (UWP) app that provides a great experience with a user's photos means you're going to have to wade deep into this chaos.

In the Window 10 Anniversary Update, Microsoft added new features to make managing this chaos easier. The system is now able to provide a list of all the changes that are happening in a library, from a picture being taken all the way up to entire folders being deleted. This is a huge help if you're looking to build a cloud backup provider, track files being moved off the device or even just display the most recent photos.

Showing the most recent photos taken by a device isn't just for developers trying to build the next Instagram. Recently I had the pleasure of working with enterprise partners as they built inspection apps for their organizations. The apps follow a similar pattern: An inspector visits a site with a Windows tablet or phone, fills in a report with some information about the site, takes pictures of the site using the device and, finally, uploads the report to a secure server. For all of the companies, it's critical that the correct, unmodified photos are uploaded with the reports.

Over the next few pages I'm going to walk through building an enterprise inspection app. Along the way I'll point out some particularly challenging issues I encountered during the development process and note how you can avoid them in your app. These lessons can also apply to any other app that's looking to track changes in the file system, such as a cloud backup service, but because inspection apps are quite common, I'll start there and you can modify the code for whatever type of application you're building.

This article discusses:

- Getting access to pictures from the camera
- Registering for foreground and background file system change notifications
- Leveraging the system to track changes while your app isn't running
- Reading and handling file system change lists from the system

Technologies discussed:

Universal Windows Platform App, Window 10 Anniversary Update

Inspections: Everybody Does Them

Across enterprises of all sizes and industries, one thing is common: There are inspectors overseeing critical business processes. From massive manufacturing companies tracking the state of their equipment to retailers ensuring that displays are assembled correctly, businesses depend on making sure things are done consistently and safely across all of their sites.

The basic process I'm going to support here is quite simple:

1. An inspector creates a new report instance for the site on his tablet.
2. The Inspector takes pictures of the site for a report.
3. The pictures are uploaded to a secure server along with the report.

In step 2, however, a number of things can go wrong:

- The inspector could pick the wrong pictures to attach to the report.
- A picture could be modified to show incorrect information.
- A picture could be accidently deleted before the report is uploaded but after the inspector leaves a location.

In any of these cases, the report would be invalid and require the inspector to repeat the inspection, an added expense for the business. Luckily, with the new change-tracking APIs in Windows 10 Anniversary Update, there's a simple way to prevent such mistakes and help users complete their tasks quickly and accurately.

Figure 1 **Types of Change Notifications**

	Foreground Change Events	Background Change Notifications
Lifespan	Only available when your app is running	Will trigger a background task even if your app isn't running
Scope	Customizable to any folders or libraries on the system	Only named libraries (pictures, video, music, documents)
Filters	Can filter to raise events only for specific file types	Will raise events for any file or folder change
Triggering mechanism	Named event	Background task trigger

The Basics: Getting Access to Pictures from the Camera

The first step is making sure the application has access to the pictures coming from the camera. In Windows, the system camera will write to the Camera Roll folder, which is a subfolder of the Pictures library. I could (and in fact did) write an entire article about how to access the Pictures library (bit.ly/2dCdj40), but here are the basics:

- Declare your intent to access the Pictures library in your manifest (bit.ly/2dqoRJX) by adding Capability Name="musicLibrary"/> in the manifest editor or checking the Pictures Library box under the Capabilities tab in the wizard.
- Get a StorageFolder representing the location where pictures from the camera are being written to using KnownFolders.CameraRoll (bit.ly/2dEiVMS).
- Get an object representing the entire pictures library using StorageLibrary.GetLibraryAsync(KnownLibraryId.Pictures) (bit.ly/2dmZ85X).

Don't worry if the device has an SD card and a user setting for writing new pictures to either internal storage or the SD card. The KnownFolder class will abstract all that away for you and give you a virtual folder encompassing all the locations to which the camera can write a file.

Getting Notified When Something Changes

Once you have access to the files, it's time to start tracking the changes. I recommend doing this as early as possible. In a perfect world, inspectors would always create a new report before they start taking pictures, but in reality they've generally taken a few pictures before they remember to create a new report.

Setting up the notifications and change tracking involves three steps:

1. Initializing the change tracker, which tells the system what libraries you're interested in tracking. The tracking will continue even when your app isn't running, and the app can read the list of changes at any time.
2. Register for change notifications in the background, which will activate your app's background task whenever there's a change in a library, whether or not it's currently running.
3. Register for change notifications in the foreground. If your app is in the foreground, you can register for additional events whenever a file is changed under a certain scope.

Note that steps 2 and 3 could potentially overlap. I'll cover how to set up both types of notifications in this article, but the chart in **Figure 1** can help you choose which you'd like to use in your app. The general recommendation is to always use background change notifications with StorageLibraryContentChangeTrigger, and to use foreground events if you have a specific UI need, such as displaying a view of the file system to your users.

The StorageLibraryChangeTracker is a new class added in the Anniversary Update (bit.ly/2dMFlfu). It allows apps to subscribe to a list of changes that are happening in a library. The system watches all the files in the library and builds up a list of the changes that happen to them. Your app can request the list of changes and process them at its leisure.

StorageLibraryChangeTracker is very similar to the NTFS change journal, if you've ever worked with that, but it works on FAT-formatted drives, as well. For more information you can read through the in-depth discussion on my blog (bit.ly/2dQ6MEK).

Initializing the StorageLibraryChangeTracker is quite simple—you just get the library-specific instance of the change tracker and call Enable:

```
StorageLibrary picsLib =
  await StorageLibrary.GetLibraryAsync(KnownLibraryId.Pictures);
picsLib.ChangeTracker.Enable();
```

From this point, the change tracker will maintain a list of all the changes to the library. Now let's make sure your app gets notified with each change.

Listening to Foreground Notifications

To listen to foreground notifications your app has to create, execute, and hold open a query over the location whose changes concern you. Creating and executing the query tells the system which locations are interesting to your app. Holding a reference to the results of that query afterwards indicates that your app wants to be notified when something changes:

```
StorageFolder photos = KnownFolders.CameraRoll;

// Create a query containing all the files your app will be tracking
QueryOptions option = new QueryOptions(CommonFileQuery.DefaultQuery,
  supportedExtentions);
option.FolderDepth = FolderDepth.Shallow;
// This is important because you are going to use indexer for notifications
option.IndexerOption = IndexerOption.UseIndexerWhenAvailable;
StorageFileQueryResult resultSet =
  photos.CreateFileQueryWithOptions(option);
// Indicate to the system the app is ready to change track
await resultSet.GetFilesAsync(0, 1);
// Attach an event handler for when something changes on the system
resultSet.ContentsChanged += resultSet_ContentsChanged;
```

As you can see, I'm using a couple of interesting optimizations that might be helpful in your app:

Figure 2 **Registering a Background Task**

```
// Check if your app has access to the background
var requestStatus = await BackgroundExecutionManager.RequestAccessAsync();
if (!(requestStatus ==
  BackgroundAccessStatus.AllowedMayUseActiveRealTimeConnectivity ||
  requestStatus == BackgroundAccessStatus.AllowedSubjectToSystemPolicy ||
  requestStatus ==
    BackgroundAccessStatus.AllowedWithAlwaysOnRealTimeConnectivity ||
  requestStatus == BackgroundAccessStatus.AlwaysAllowed))
{
  log("Failed to get access to the background");
  return;
}
// Build up the trigger to fire when something changes in the pictures library
var builder = new BackgroundTaskBuilder();
builder.Name = "Photo Change Trigger";
StorageLibrary picturesLib =
  await StorageLibrary.GetLibraryAsync(KnownLibraryId.Pictures);
var picturesTrigger = StorageLibraryContentChangedTrigger.Create(picturesLib);
// We are registering to be activated in OnBackgroundActivated instead of
// BackgroundTask.Run; either works, but I prefer the single-process model
builder.SetTrigger(picturesTrigger);
BackgroundTaskRegistration task = builder.Register();
```

- CommonFileQuery.DefaultQuery makes the entire operation much faster in cases where the indexer isn't available. If another sort order is used and the indexer isn't available, the system must walk the entire query space before it returns the first result.
- The query is a shallow query. This works because the camera will always write to the root of the camera roll, and avoiding a deep query minimizes the number of files the system has to track for changes.
- Using the indexer isn't mandatory, but it speeds up notifications appearing in the app. Without the indexer, notifications can take up to 30 seconds to reach your app.
- Querying for one file is the fastest way to start tracking changes in an indexed location, though you may want to query for more files in case the UI needs them.

Now anytime an item in the query changes, the event handler will be triggered, giving your app a chance to process the change.

Registering for the Background Change Trigger

Not all changes are going to happen while your app is in the foreground, and even if your app is in the foreground it may not need the granularity of foreground notifications. StorageLibraryContentsChangedTrigger is a great way to be notified when anything changes in a library. Because you register a background task using the standard process (bit.ly/2dqKt9i), I'll go through it quickly (see **Figure 2**).

There are a couple of important things to note in the sample in **Figure 2**.

You can still use the old two-process model to register your background task, but take a good look at the single-process model that was added in the Anniversary Update. It quickly won me over with how simple it is to use and how easy it is to receive background triggers while your app is in the foreground.

StorageLibraryContentChangedTrigger will fire for any changes in the Pictures library, which may include files that aren't interesting to your app. I'll cover how to filter those out in a later section, but it's always important to be aware that sometimes there will be nothing to do when your app is activated.

It's worth checking your background task whether you're running in the background or foreground because the resource allocations are different. You'll find more detail on the resource allocation model at bit.ly/2cNvcSr.

Figure 3 **Checking for Changes to Files**

```
foreach (StorageLibraryChange change in changes)
{
  if (change.ChangeType == StorageLibraryChangeType.ChangeTrackingLost)
  {
    // Change tracker is in an invalid state and must be reset
    // This should be a very rare case, but must be handled
    picturesLib.ChangeTracker.Reset();
    return;
  }
  if (change.IsOfType(StorageItemTypes.File))
  {
    await ProcessFileChange(change);
  }
  else if (change.IsOfType(StorageItemTypes.Folder))
  {
    // No-op; not interested in folders
  }
  else
  {
    if (change.ChangeType == StorageLibraryChangeType.Deleted)
    {
      UnknownItemRemoved(change.Path);
    }
  }
}
// Mark that all the changes have been seen and for the change tracker
// to never return these changes again
await changeReader.AcceptChangesAsync();
```

Reading the Changes

Now your app is going to be given a chance to run code every time something changes in the camera roll, either in the foreground or the background. To figure out what has changed, you need to read the set of changes from StorageLibraryChangeTracker. The first step is to get a reader object, which will allow you to enumerate the changes that have happened since the last time your app checked. While you're at it, you can also grab the first batch of changes to process:

```
StorageLibrary picturesLib =
  await StorageLibrary.GetLibraryAsync(KnownLibraryId.Pictures);
StorageLibraryChangeTracker picturesTracker= picturesLib.ChangeTracker;

picturesTracker.Enable();
StorageLibraryChangeReader changeReader = picturesTracker.GetChangeReader();
IReadOnlyList<StorageLibraryChange> changes = await changeReader.ReadBatchAsync();
```

Once you get a set of changes, it's time to process them. In this app, I'm going to focus only on pictures being changed and ignore all other file and folder changes. If you're interested in other types of changes on the system, my deep dive on the change tracker has the details about all the different types of changes (bit.ly/2dQ6MEK).

I'm going to walk through the changes and pull out the ones in which I'm interested. For this application, it's going to be any change to the contents of a .jpg file, as shown in **Figure 3**.

Here are a few interesting things about the code snippet in **Figure 3**.

The first thing I do is check for StorageLibraryChangeType.ChangeTrackingLost. This should happen only after a large file system operation (where the system doesn't have enough storage for the entire set of changes) or in the case of a critical internal failure. In either case, anything being read from the change tracker can no longer be trusted. The app must reset the change tracker for future results to be trustworthy.

UnknownItemRemoved(change.Path) will be hit any time a file or folder is deleted from a FAT partition, such as an SD card. Once an item

Extreme Performance Linear Scalability

NCache™

Distributed Cache

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

NosDB

NoSQL Database

- Schema-Free JSON Documents
- Multiple Shards & Data Replication
- Powerful SQL, LINQ, ADO.NET

100% Native .NET

Open Source (Apache 2.0 License)

Microsoft Azure amazon web services™ Windows Server 2012 Visual Studio

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

is deleted from a FAT partition the system can't tell whether it was a directory or a file that was deleted. I'm going to walk through some code in a bit that shows how you can figure out what happened in your app.

When all the changes have been processed, you call changeReader.AcceptChangesAsync. This tells the change tracker that your app has handled all the changes and doesn't need to see them again. Next time you create a StorageLibraryChangeReader it will contain only changes that have happened since this point. Not calling AcceptChangesAsync will cause the internal change buffer to fill up and overflow, resulting in StorageLibraryChangeType.ChangeTrackingLost.

Handling a Change

Now that you know how to walk through the changes, the next step is to flesh out the ProcessFileChange method. One key concept to remember is that opening files (by creating StorageFile objects) is extremely expensive. To get a StorageFile object in your process, the system must make a cross-process call to check permissions, create a handle to the file, read some metadata about the file from the disk, and then marshal the handle and metadata back to your app's process. Thus, you want to minimize the number of StorageFiles you create, especially if you don't intend to open the file streams.

The change tracker provides your app with paths and the type of change to help you determine whether a file is of the type your app is interested in before you create the StorageFile. Let's start by doing as much filtering as possible with this information before creating the StorageFile, as shown in **Figure 4**.

Let's break down what's happening here by looking at each of the case statements.

> One key concept to remember is that opening files by creating StorageFile objects is extremely expensive.

New File in the Library: In the first statement I'm looking for new files that have been added to the library. In the case of the system camera, the file is going to be created in the library and I'll see a change of type StorageLibraryChangeType.Created. Some third-party apps actually create the image in their app data container and then move it to the library, so I'm going to treat StorageLibraryChangeType.MovedIntoLibrary as a creation, too.

I'm going to process StorageLibraryChangeType.MovedOrRenamed as a new file. This is to work around an oddity of the built-in camera on Windows phone. When the camera takes a picture, it writes out a temporary file with the ending .jpg.~tmp. Later it will finalize the picture, removing the .~tmp extension. If your app is quick enough to catch the file before the camera is done finalizing the image, it may see a rename event instead of the creation event.

Because this app is going to be interested only in pictures taken by the user, I'll filter down to only files with the .jpg extension. You could do this by creating a StorageFile and checking its ContentType property, but I'm trying to avoid creating unneeded StorageFiles. Once I know a file is one I'm interested in, I'm going to hand the file off to another method to do some data processing.

I'm using a cast here instead of the as keyword because I've already asked StorageLibraryChange what type the StorageItem will be using: StorageLibraryChange.IsOfType(StorageItemTypes.File).

File Removed from Library: In the second case, the app is looking at situations where a file has been removed from the library. You'll note that I've grouped two different change types together again. StorageLibraryChangeType.Deleted will be raised whenever a file is deleted permanently from disk, which is the classic case of deletion using the file system APIs. However, if the user manually deletes the file from File Explorer instead, the file will be sent to the Recycle Bin. This shows as StorageLibraryChangeType.MovedOutOfLibrary because the file is still resident on disk.

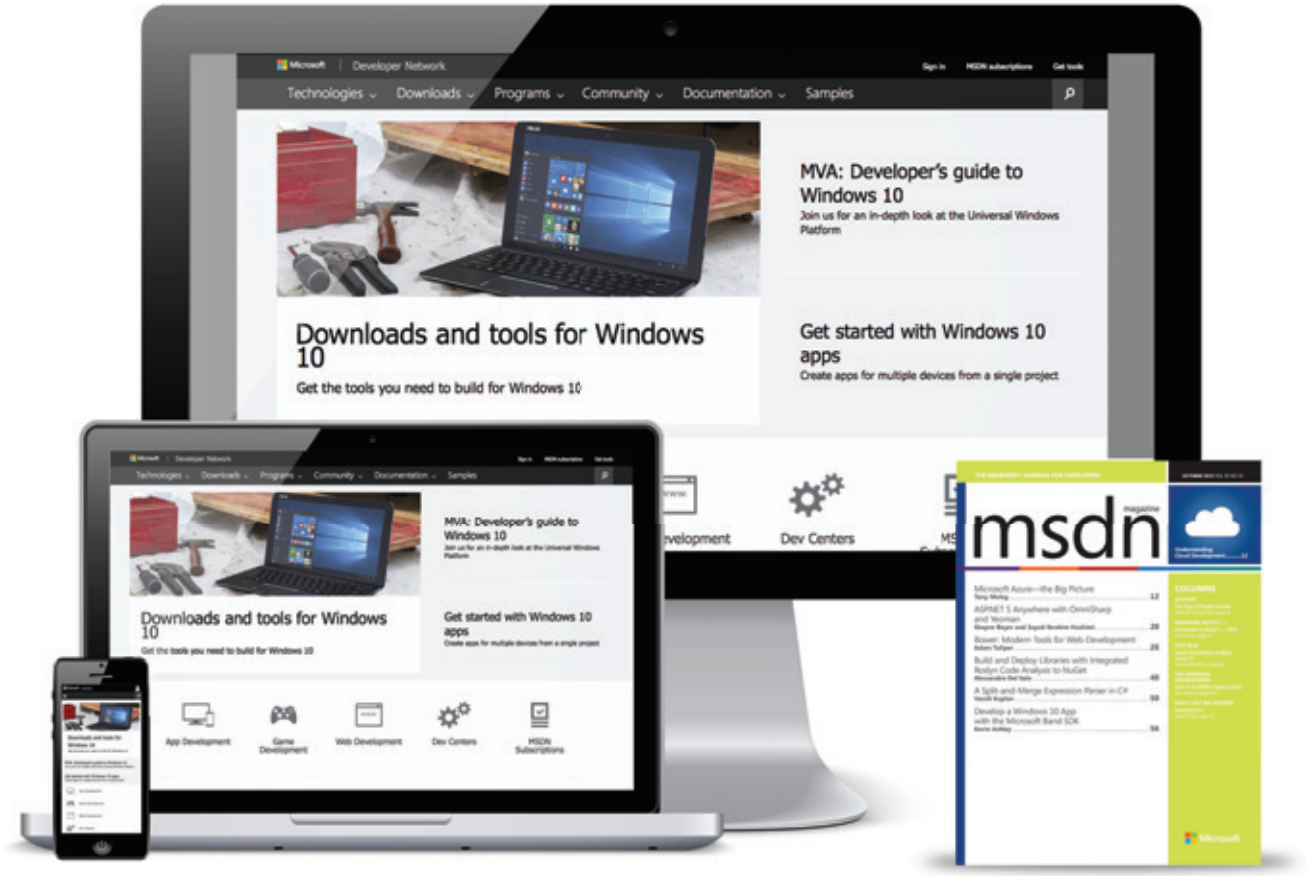
In this case, I'm going to raise a warning to the inspector that the file is gone, in case it was an accidental deletion. In more security-conscious applications, it might make sense to store file deletions or modifications for later auditing in case of an investigation.

Figure 4 **Processing Changes**

```
private async Task ProcessFileChange(StorageLibraryChange change)
{
  // Temp variable used for instantiating StorageFiles for sorting if needed later
  StorageFile newFile = null;
  switch (change.ChangeType)
  {
    // New File in the Library
    case StorageLibraryChangeType.Created:
    case StorageLibraryChangeType.MovedIntoLibrary:
    case StorageLibraryChangeType.MovedOrRenamed:
      if (change.Path.EndsWith(".jpg", StringComparison.OrdinalIgnoreCase))
      {
        StorageFile image = (StorageFile)(await change.GetStorageItemAsync());
        AddImageToReport(image);
      }
      break;
    // File Removed From Library
    case StorageLibraryChangeType.Deleted:
    case StorageLibraryChangeType.MovedOutOfLibrary:
      if (change.Path.EndsWith(".jpg", StringComparison.OrdinalIgnoreCase))
      {
        var args = new FileDeletedWarningEventArgs();
        args.Path = change.Path;
        FileDeletedWarningEvent(this, args);
      }
      break;
    // Modified Contents
    case StorageLibraryChangeType.ContentsChanged:
      if (change.Path.EndsWith(".jpg", StringComparison.OrdinalIgnoreCase))
      {
        newFile = (StorageFile)(await change.GetStorageItemAsync());
        var imageProps = await newFile.Properties.GetImagePropertiesAsync();
        DateTimeOffset dateTaken = imageProps.DateTaken;
        DateTimeOffset dateModified = newFile.DateCreated;
        if (DateTimeOffset.Compare(dateTaken.AddSeconds(70), dateModified) > 0)
        {
          // File was modified by the user
          log("File path: " + newFile.Path + " was modified after being taken");
        }
      }
      break;

    // Ignored Cases
    case StorageLibraryChangeType.EncryptionChanged:
    case StorageLibraryChangeType.ContentsReplaced:
    case StorageLibraryChangeType.IndexingStatusChanged:
    default:
      // These are safe to ignore in this application
      break;
  }
}
```

msdn magazine

Where you need us most.



Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

LIVE! 360
TECH EVENTS WITH PERSPECTIVE

MSDN.microsoft.com

Modified Contents: The contents of a file being modified is an interesting case for this app. Because this app might be used for safety inspections, you don't want to allow users to change the images before they're uploaded to the report, though there may be valid reasons for the contents of an image to be changed after a picture is taken.

You might see a notification of the type StorageLibraryChangeType.ContentsChanged being raised about three to 60 seconds after a picture is taken with the camera. This is because it can sometimes take up to a minute to get the coordinates from the GPS system. In this app I'm not concerned about GPS information so I'm processing the file right away. In some apps it might make sense to check if the location data has been written to the file, and if it hasn't, to wait until the location is determined.

> Experiment with your configuration! This is the most important piece of advice I can give you about creating an app that tracks file system changes.

In this case, I'm going to go for a very safe middle ground. If a file is modified more than 70 seconds after it was taken, I'm going to assume it was modified by the user and log out a compliance issue for later investigation. If it was modified within the 70-second window, I'll assume this was done by the system when adding the GPS data and can be safely ignored.

Encryption Changed: Whether you should care about this case comes down to one question: Does your enterprise use Windows Information Protection (WIP) with encryption to protect its sensitive information? If so, then this type of change is a huge concern. It means that someone has changed the protection on the file, and the file could be sitting on the disk unprotected. Walking through all of the WIP information about how to check if the file is secure (versus just being moved to another protection level) is beyond the scope of this article. If you're deploying WIP in your enterprise, this type of change is important for you to monitor.

For those not using WIP or encryption to protect sensitive corporate assets, I highly recommend you look into it, but for now this change is safe to ignore.

Ignored Cases: Finally, in this application it makes sense to ignore some of the things that could be happening on the file system. StorageLibraryChangeType.IndexingStatusChanged only matters in the case of desktop apps that are going to be doing repeated queries of the library and expecting the same results each time. StorageLibraryChangeType.ContentsReplaced indicates that a hard link for the file has been changed, which isn't interesting for this application.

Detecting Deletion of an Image on an SD Card

As you'll recall, the system can't determine if a removed item is a file or folder after it's been deleted on FAT drives such as an SD card. Thus, some special sorting code is needed if you care about deletions on FAT drives. For my app, I only want to know if pictures are being deleted from the SD card, which makes the code really simple:

```
private void UnknownItemRemoved(StorageLibraryChange change)
{
  if (change.Path.EndsWith(".jpg", StringComparison.OrdinalIgnoreCase))
  {
    var args = new FileDeletedWarningEventArgs();
    args.Path = change.Path;
    FileDeletedWarningEvent(this, args);
  }
}
```

The code simply checks to see if the item used to have a .jpg extension, and if so raises the same UI event as in **Figure 4**.

Final Thoughts

Once the basic code in place, there are a couple of things worth considering that can help make sure your new app will run as smoothly as possible.

Experiment with your configuration! This is the most important piece of advice I can give you about creating an app that tracks file system changes. Write a quick app that logs what's happening on your devices before you get too deep into coding. A lot of components try to get away with writing out temp files and trying to quickly delete them, such as the built-in camera I discussed earlier. I've even seen cases in the wild where apps were writing out incomplete files, closing the handle, then immediately reopening the handle to finish writing out the file to disk. With a change-tracking app, you're not only going to start seeing all of these changes, but potentially get in the middle of these operations. It's important to be a good citizen of the system, which means understanding and respecting what other apps are trying to do.

Keep in mind that because SD cards can be removed from the device while it's powered off, and there's no journaling on FAT-based file systems, change tracking can't be guaranteed on an SD card across boot sessions. If your enterprise has very strict requirements about ensuring that files are not tampered with, consider using a mobile device management policy to force encrypted camera images to be written to internal storage only. This will ensure that the data is protected at rest and that all changes to the file are accounted for.

Wrapping Up

And that's it. The inspection app is now ready to automatically add files when they're created on the system, and to notify users when one is deleted. Although it may seem simple, enabling this experience allows users to focus on their inspections instead of picking thumbnails out of the system-picker experience. If the inspector is constantly taking pictures of similar-looking equipment, automatically including recent images in the report can greatly cut down on mistakes that cost businesses time and money. Most important, it can help your app stand out among the crowded field of enterprise apps for Windows. ■

Adam Wilson *is a program manager on the Windows Developer Ecosystem and Platform team, working on the Windows indexer and push notifications. You can reach him at Adam.D.Wilson@microsoft.com.*

Thanks to the following Microsoft technical experts who reviewed this article: *Brendan Flynn, Mary Anne Noskowski and Kyle Yuan*

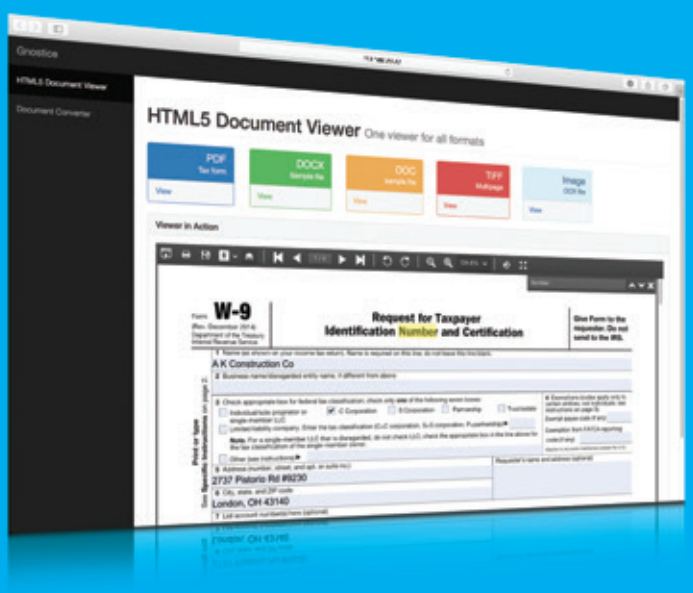
Document Technology for Everybody

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

Use Bot Framework for Anytime, Anywhere Access to Application Data, Part 2

Srikantan Sankaran

In the last issue I introduced the Microsoft Bot Framework, and discussed how you can take advantage of it to meet a challenge that many organizations face today, which is to give customers interactive access to their services, via voice and text (msdn.com/magazine/mt788623). In the article I described a business scenario that involved consolidating information from different line-of-business applications deployed at an insurance provider organization, using technologies like Microsoft Flow and Azure Logic Apps. The content, comprising both structured and unstructured data, is indexed using Azure Search, and then made available for client applications to consume.

This article is based on a preview version of the Microsoft Bot Framework; all information is subject to change.

This article discusses:

- Creating and training the LUIS model for different scenarios
- Creating the bot application
- Adding Azure Search to the Visual Studio solution
- Identifying the bot user and storing the user profile in the bot state
- Deploying the bot to Azure and enabling it across channels

Technologies discussed:

Visual Studio 15, Microsoft Bot Framework, Azure Search, LUIS

Code download available at:

bit.ly/2cOfANh

Figure 1, which also appeared in the previous article, depicts the architecture of the solution that implements the scenario.

In this article, I'm going to build and deploy a bot application that can consume the information gathered via Azure Search. The bot application will be deployed as an Azure Web app, and published through the Microsoft Bot Framework Connector Service. Through the Connector Service, the Bot application will be accessed from the Skype channel, which will give consumers access to the status of their insurance policy requests and let them download the issued policy documents, schedule site inspection visits and so on. Consumers will be able to identify themselves to the bot using the Microsoft account they registered with when applying for an insurance policy, and can use either messaging on the Skype client or voice commands to interact with the bot. The bot application integrates with the LUIS service, one of several services available in Azure Cognitive Services, to interpret conversations from the consumer in order to execute queries on Azure Search.

The steps required to build the solution are explained in the following sequence. The snippets of code are taken from the Visual Studio 2015 solution, which can be downloaded with this article. Here's what you should do if you're following along:

- Create and publish a LUIS model that captures the different conversations that need to be supported in the bot, using the UI provided by the LUIS service.
- Use the Visual Studio 2015 template to create a new bot application. You can then either add the code blocks from the solution provided with this article, or follow the rest of

Visual Studio MAGAZINE

READER'S CHOICE AWARD WINNER 2016

DevExpress®

Visual Studio MAGAZINE 1 READER'S 2016 CHOICE

Thank You x 18

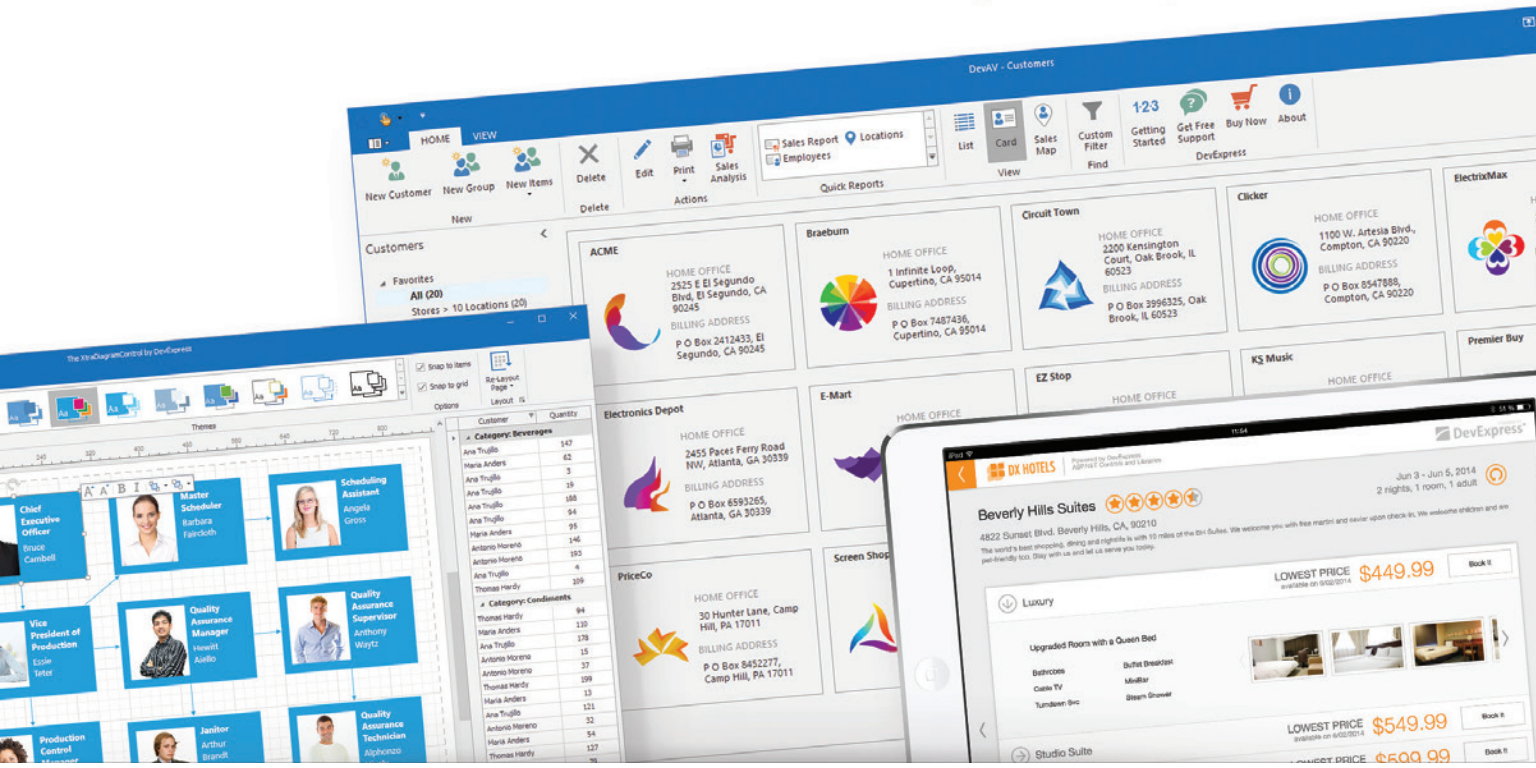
Words alone cannot express our sincere gratitude for your steadfast support. For 2 consecutive years, you've honored us with 18 VSM's Readers' Choice Awards. We thank you for your continued patronage and for the faith you've placed in DevExpress over the years.

Since 1998, our unshakeable mission has been to create best-of-breed software development tools so you can deliver high-impact business solutions across your enterprise. We remain committed to this simple proposition and are working hard to perfect our product line so you can meet tomorrow's challenges head-on.

Please remember that we are here to help. Write to us anytime with thoughts, opinions and feedback.

Thank you once again from all of us at DevExpress
management@devexpress.com

WIN WPF ASP MVC HTML



To learn more, please visit our website → www.devexpress.com

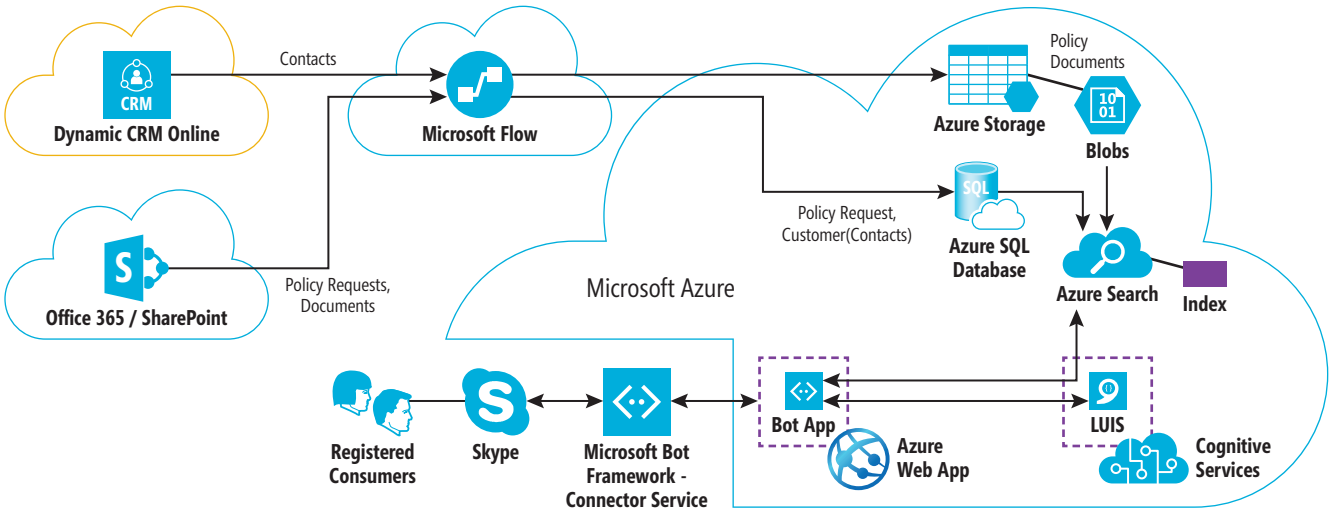


Figure 1 **Architecture of the Solution**

the article referring to the downloaded solution file. The MessageController.cs file is the entry point into the bot application, which is a special type of a .NET MVC application.

- Embed the LUIS functionality into the bot, using the LUIS dialog available in the Bot Framework SDK. This is implemented in the PolicyInfoDialog.cs file in the solution.
- Using the Azure Search SDK, implement code that invokes the Azure Search APIs and execute the search requests interpreted by the LUIS service. This is implemented in the SearchService.cs file in the solution.
- Implement a FormFlow dialog to capture the user's Microsoft account in the Skype channel, in an interactive way, when a user request is made the first time. This is implemented in the SigninForm.cs file in the accompanying solution.
- Use the Azure Search SDK to retrieve the user profile information based on the Microsoft account captured when the user first accesses the bot. This is also implemented in SearchService.cs
- Use the Bot State Service to retain the User Profile information. The bot user's address and contact information, which is stored in the Bot State Service, is used when scheduling a site inspection visit. This is also implemented in the PolicyInfoDialog.cs file.
- Test the bot application by running it locally as a service on your development machine, and use the Bot Framework Channel Emulator to assess user interaction with the service.
- After testing the bot application locally, deploy it to a Web app in Azure right from Visual Studio.
- Register the bot application with the Microsoft Bot Framework Connector Service and configure in it the URL of the bot application deployed to Azure.
- Add the bot application name, app ID and secret information obtained during registration of the bot into the Visual Studio solution and republish it to Azure.
- Enable the bot application for access from channels like Skype and Slack, from the bot developer Portal.

You can download the Visual Studio 2015 solution for the bot application, and a file export of the LUIS model used in this application from the GitHub repository at bit.ly/2cOfANh.

Creating and Training the LUIS Model for Different Use-Case Scenarios

Figure 2 shows how I mapped out how a user could potentially converse (via a LUIS utterance) with the bot and how this should be interpreted as a specific LUIS intent or action to be performed via Azure Search, and how to extract keywords (LUIS entities) from the LUIS utterance to be used as query parameters when performing Search.

Figure 2 **Mapping Utterances and Intents**

Utterances or Conversation	Mapped Intents or Actions	Qualifiers or Parameters
Something LUIS can't interpret (or isn't trained for).	None (Default)	Not applicable
Get (or show) all my policy requests.	GetPolicyRequests	Implied for logged-in bot user
I have a policy request application, *or* is my policy request approved?	GetPendingPolicyRequests	Implied for policy requests where the status is other than approved
Get me the details of policy request number VehPolicy001.	GetPolicyRequestDetails	Policy Request Id – VehPolicy001
Get the status of my life insurance policy request.	GetPolicyRequestStatusByType	Policy Request Type – Life
Get policy document number Policy0101 issued this year.	GetPolicyDocument	File Name – Policy0101 and lastmoddate – current year
Schedule a site-inspection visit tomorrow.	ScheduleSiteVisit	Inspection Date – Tomorrow

The LUIS service provides a Web application at luis.ai in which a LUIS application model can be created visually for the problem at hand. **Figure 3** displays the LUIS model configured for this scenario.

The intents, entities and utterances identified in the previous step can all be created in the LUIS application. Use the Intents menu

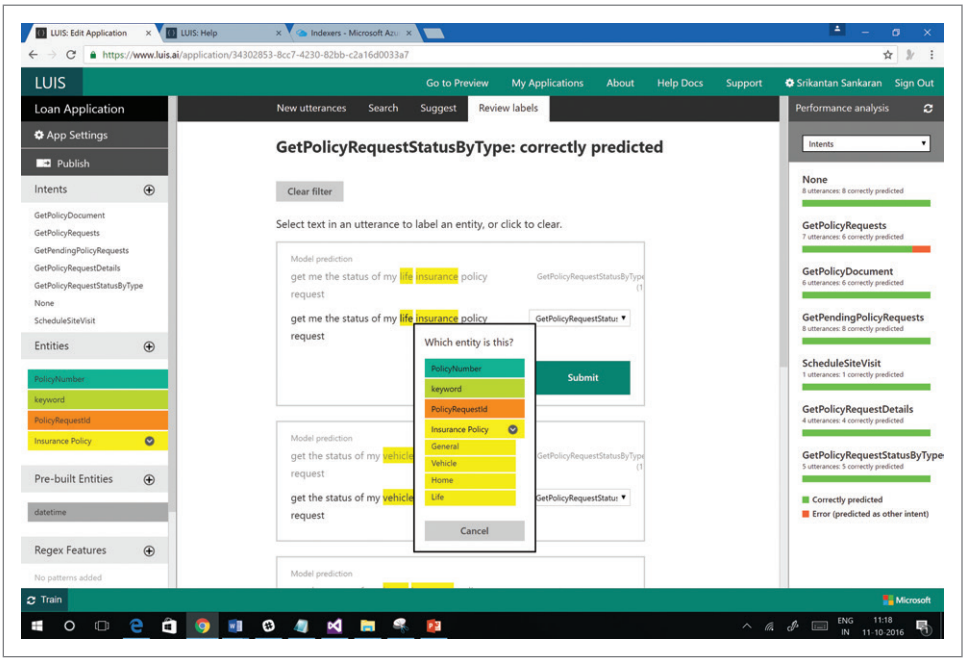


Figure 3 **The LUIS Model for My Scenario**

to add the intents and the Entities menu to create the entities. Note that there are two special kinds of entities:

- **Hierarchical entities:** In this case, the "Insurance Policy" is created as a hierarchical entity, the parent, with the various types, Life, Vehicle, Home and so on, created as the children.
- **Pre-built entities:** I used a pre-built, LUIS-provided entity called Datetime. Input parameters such as the year a policy document was generated, or the date when a site inspection visit is scheduled, will be mapped by the LUIS engine to these pre-built entity types.

The entity with the name "Keyword" will be used to perform a full text search in the policy documents index in Azure Search.

Now, select the New utterances tab and create utterances for each of the conversations identified previously, and label them.

For each utterance, identify or label the intent and the associated entities. For example, In **Figure 3**, I mapped the utterance, "get me the status of my life insurance policy request," to the intent "GetPolicyRequestStatusByType," and mapped the text "Life Insurance" to the hierarchical entity "Insurance:life."

Repeat this step to add other variations, such as when the policy type is Vehicle, Home or General. Also, consider variations where the same request could be made in a different way. However, you don't have to capture every permutation and combination of ways the same message might be conveyed. For example, when I add a new utterance with the phrase, "show me the status ...," as shown in **Figure 4**, LUIS could rightly interpret that as a variant of the previous one, which used "get me the status ...." LUIS made the right suggestions on the intent and the associated entities, even though the question was asked in a different way.

After each utterance is captured, the LUIS application provides the option to Train the Model. This results in the AI engine in LUIS generating optimized models that can then be used by client applications. This step should be repeated whenever new utterances are labeled or changes are made to existing ones.

Publishing the LUIS model allows client-side applications to invoke it, using a REST API. The publish dialog in the application also lets you test the model to ensure it works as expected. **Figure 5** shows how to quickly test the LUIS model by passing in a conversation phrase and viewing the output in the browser. (For convenience, **Figure 5** shows the snapshots from three different invocation of utterances and the corresponding results side by side.)

Notice that the intent recommended by LUIS has the maximum probability value assigned to it, and appears first in the list. The entities that were extracted from the phrase are also returned in the response. If there are multiple entities identified in the utterance, they're returned, as well. You can see how the LUIS model interprets terms like "this year" in the utterance phrase, maps it to a pre-built entity "Datetime" and returns the value "2016," which the client application can directly use.

Creating the Bot Application

Now that the LUIS model is ready, it can be used in a bot application. Download the Visual Studio 2015 Template from aka.ms/bf-bc-vstemplate to create the application. (Visit bit.ly/2dYrwyW to learn the basics of building a bot application and how to use a bot emulator to build and test the bot locally.)

The project created when using this template is a variant of an MVC project. The entry point to the project is MessageController.cs, which processes incoming messages from channels like Skype or Slack, and sends back the responses.

The LUISDialog for the Bot Framework is used to implicitly pass messages from the bot to the LUIS model. For each of the intents implemented in the LUIS model, a corresponding method is implemented that can be called directly when the bot

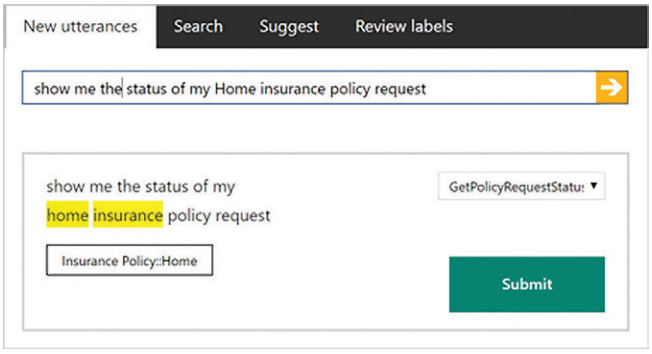


Figure 4 **Labeling LUIS Utterances and Handling Variations**

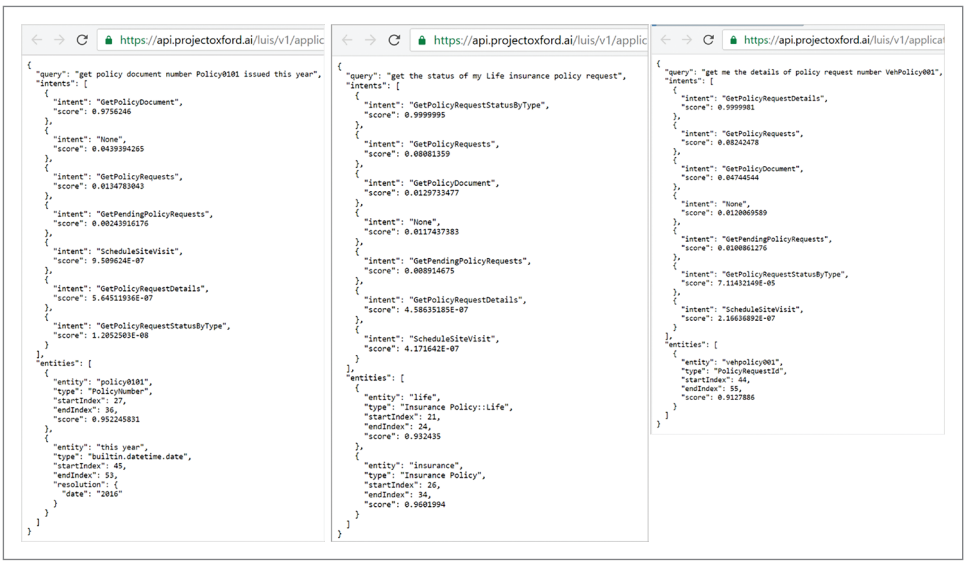


Figure 5 **Testing the LUIS Model**

application is running. This obviates the need for custom code to call the LUIS REST APIs to interpret the conversation and to identify the intent and entities.

The LUIS application Id and the secret values are used to decorate the class-level attributes, namely LuisModel. The LuisIntent attribute is used on the method that provides the implementation when that intent is encountered in the user request. This is all that's required to get the LUIS integration going in the bot. **Figure 6** depicts how this is implemented in code.

At this point, the implementation in each of the methods can be simple; I can simply return a message identifying which LuisIntent implementation was invoked.

In MessageController.cs, add the following snippet to capture the incoming message from the bot user and hand it off to the LuisDialog:

```
if (activity.Type == ActivityTypes.Message)
  {
    await Conversation.SendAsync(activity, () => new PolicyInfoDialog());
  }
```

Run the bot application, launch the Bot Framework Channel Emulator, which can be downloaded from aka.ms/bf-bc-emulator. Type in the utterances you trained the LUIS Model for earlier, and ensure that the response message indicates that the correct method mapped to the LUIS intent was invoked.

Figure 6 **Implementing LUIS Integration**

```
[LuisModel("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx", "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx")]
[Serializable]
  public class PolicyInfoDialogs : LuisDialog<object>
  {

    [LuisIntent("")]
    public async Task None(IDialogContext context, LuisResult result)
    {
      // Called when the incoming utterance is not recognized (default bin)
    }

    [LuisIntent("GetPolicyRequests")]
    public async Task GetPolicyRequests(IDialogContext context, LuisResult result)
    {
      //
    }
        ----------------------
      }
```

Search Integration

Now I'll add the Azure Search SDK for .NET to the Visual Studio solution using the NuGet Package Manager. The SearchService.cs file in the solution uses this SDK and encapsulates all integration with the Azure Search Service. It implements methods that execute queries on Azure Search, return the UserProfile of the customer and retrieve PolicyRequests and PolicyDocuments.

Search commands for the different scenarios are built using the Lucene query syntax, which is implemented in the PolicyInfoDialog.cs file. Here are some examples:

Retrieve all policy requests for the bot user in the conversation:

```
command += "CustomerId:(" + msftid + ")";
```

Retrieve all policy requests for the bot user that haven't yet been approved:

```
command += "CustomerId:(" + msftid + ") -PolicyStatus:(Approved)";
```

Query policy requests for the bot user based on the policy request Id:

```
public async Task GetPolicyRequestDetails(IDialogContext context, LuisResult result)
  {
    ----------------------
      string  command += "CustomerId:(" + msftid + ") AND PolicyRequestId:(" +
        result.Entities[0].Entity + ")";
    ----------------------
  }
```

The preceding snippet shows how the PolicyRequestId Entity (the input parameters to execute the search queries) is passed by the LUIS engine through LuisResult (result).

Figure 7 **Implementing Property-Based Search and Full-Text Search for Documents**

```
foreach (EntityRecommendation curEntity in request.Entities)
  {
    switch (curEntity.Type)
      {
        case "PolicyNumber": // Look for documents with this filename
          {
            command += " filename:(" + request.Entities[0].Entity + ") ";
            break;
          }
        case "keyword": // Search within documents for this keyword expression
          {
            command += " " + request.Entities[0].Entity + " ";
            break;
          }
        case "builtin.datetime.date":
        // Retrieve based on the year a document was generated
          {
            char[] delimiterChars = { '.', ':' };
        List<string> allvalues = curEntity.Resolution.Values.ToList<string>();
             string val = allvalues[0];
             string[] vals = val.Split(delimiterChars);
             command += " lastmoddate:(" + vals[0] + ") ";
             break;
          }
        default:
          {
            break;
          }
        }
    }
```

Visual Studio MAGAZINE

READER'S CHOICE AWARD WINNER 2016

LEADTOOLS®
THE WORLD LEADER IN IMAGING SDKs

Build Better Applications with LEADTOOLS Imaging SDKs

Developers know the value of a quality toolkit, and LEADTOOLS delivers. With over 25 years of experience, LEAD's comprehensive document (OCR, Barcode, PDF, Forms Recognition (ID, Passport, Invoice), Viewers, Annotations, File Formats), medical (DICOM, PACS, and HL7), and multimedia imaging SDKs. These features, available for .NET, C++, Linux, iOS, macOS, Android, and HTML5/JavaScript, offer developers more than any toolkit on the market.

LEADTOOLS literally puts millions of lines of code at your fingertips and cuts months, if not years off of your development life cycle. Whatever your programming needs, LEADTOOLS provides the best and most diverse imaging technology available.

Download a free, fully functional evaluation SDK → www.leadtools.com

Figure 7 shows how to implement both property-based search and full-text search for documents.

Note that you may want to refer to my previous article in order to relate to the property names and values in the **Figure 7** code. Also, you can download the schema of all the tables used in the database accompanying this article using the links provided in later sections.

In the Bot registry, you can enable the bot on other channels.

Formatting the Response to the User

You can format the text in the response message returned by the bot for style, to add hyperlinks or media content and so forth. This helps make the bot application more user-friendly. Here are a couple of examples …

To set the font style to bold, only for the dynamic data returned from Azure Search, wrap it inside of double asterisks (**):

```
foreach (SearchResult<PolicyRequest> result in eachGroup)
  {
    searchResponse += (counter) + ". **" + result.Document.PolicyRequestId + "**" +
      " on the insured :**" + result.Document.InsuredIdentifier + "**
      for an amount of :**" +
    result.Document.Currency + " " + result.Document.AssessedValue + "**
      has status: **" +
    result.Document.PolicyStatus + "**\n\n";

    counter++;
  }
```

To add a hyperlink to a policy document returned by Azure Search, add the Link attribute to the URL of the document:

```
foreach (SearchResult<PolicyDocument> result in results)
 {
   searchResponse += counter + ". Link: [" + result.Document.filename + "](" +
     result.Document.fileurl + ") \n";
   counter++;
 }
```

Identifying the Bot User

The Bot Framework provides the user's channel Id and the display name from the conversation context, but doesn't provide the username used to log into the channel, such as Skype. Hence, this information should be explicitly obtained when a user starts to interact with the bot. The FormFlow dialog described earlier is used to prompt the user for this information. The user profile information is retrieved from Azure Search based on the account name.

To ensure that the user interacting with the bot is the owner of the Microsoft account name shared in the FormDialog, you can integrate an interactive login to the Microsoft Account Sign-in page, and provide access to the bot only after authentication and validation. This is out of scope of the article. Instead, you simply prompt the user for the account name using the SigninForm dialog and assume the user is who he claims to be:

Figure 8 **Storing the User Profile**

```
private async Task SignUpComplete(IDialogContext context,
IAwaitable<SigninForm> result)
  {
-----------------------------------------------
-----------------------------------------------
  UserProfile profile =
    await LoadBotUserState(profileIdentityForm.MicrosoftAccount);
  if (profile == null)
  {
  message = $"Sorry, I could not locate your profile in our system.
    Please retry with the right Microsoft Account";
  }
  else
  {
    context.PrivateConversationData.SetValue<bool>("ProfileComplete", true);
    context.PrivateConversationData.SetValue<string>(
      "FirstName", profile.FirstName);
    context.PrivateConversationData.SetValue<string>("LastName", profile.LastName);
    context.PrivateConversationData.SetValue<string>("Address", profile.Address);
    context.PrivateConversationData.SetValue<string>("Phone", profile.Phone);
    context.PrivateConversationData.SetValue<string>(
      "CustomerId", profile.CustomerId);
    message = $"Thanks {profile.LastName},{profile.FirstName}
      for identifying yourself! \n\n
      Let me know how I could assist you.";
  }
    await context.PostAsync(message);
  }

    context.Wait(MessageReceived);
  }
```

```
public static IForm<SigninForm> BuildForm()
  {
    return new FormBuilder<SigninForm>()
      .Message("I would need some information before we get started.")
      .Field(nameof(MicrosoftAccount))
      .Build();
  }
```

Storing the User Profile in the Bot State

Once the user is identified during the commencement of a conversation in the bot, a user profile object is retrieved from Azure Search. This information is stored in the bot state, to be used, for example, when the user requests that a site inspection visit be scheduled. The address and contact number of the user is verified during the confirmation of the inspection visit.

The Bot Framework provides an API to store this information in the privateConversationData context, per user and conversation, as shown in **Figure 8**.

For every request, the bot application checks with the bot state

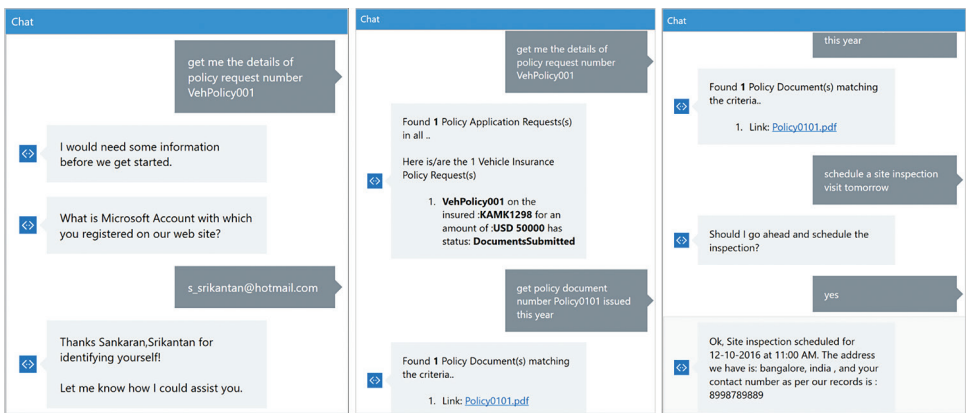


Figure 9 **Running the Bot from an Emulator**

to ensure that the identity of the user issuing the request is verified and the user profile information is available.

Using the Bot Channel Emulator Locally

The bot application requires the Azure Search namespace, access key and the index name to be used to execute the requests from the user. These parameters are stored in the SearchParameters.cs file, and their values set from the web.config file when the bot application is run. Now you can run the Visual Studio solution.

Launch the Bot Channel Emulator, and enter the conversations you trained the LUIS model for. A typical user interaction scenario is captured in **Figure 9**. Once the bot application runs successfully in the local environment, it can be published to an Azure Web app.

Deploying the Bot to Azure

Create a Web app in Azure from the Azure Portal, then, in the Application settings, add the appSetting keys and values from the web.config file in the Visual Studio solution.

Before publishing the bot application to an Azure Web app, you need to register this bot at dev.botframework.com/bots/new. This process generates a Bot ID, a Microsoft App ID and a Secret. Add these values to the bot application's web.config in Visual Studio 2015, and republish this app to Azure.

During the registration process, for the redirect URL parameter, use the SSL-enabled URL of the bot application in Azure and suffix that with "/api/messages."

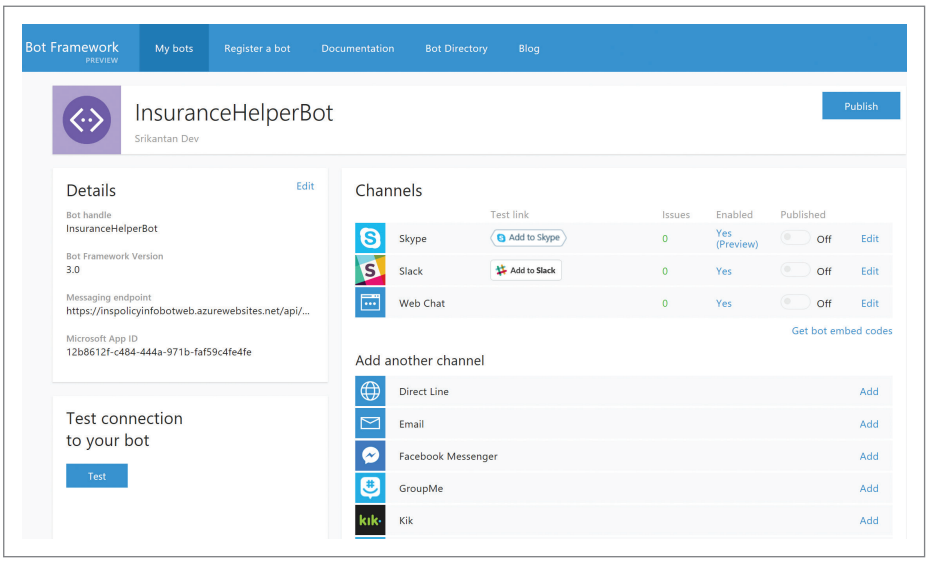


Figure 10 **Registering the Bot on Multiple Channels**

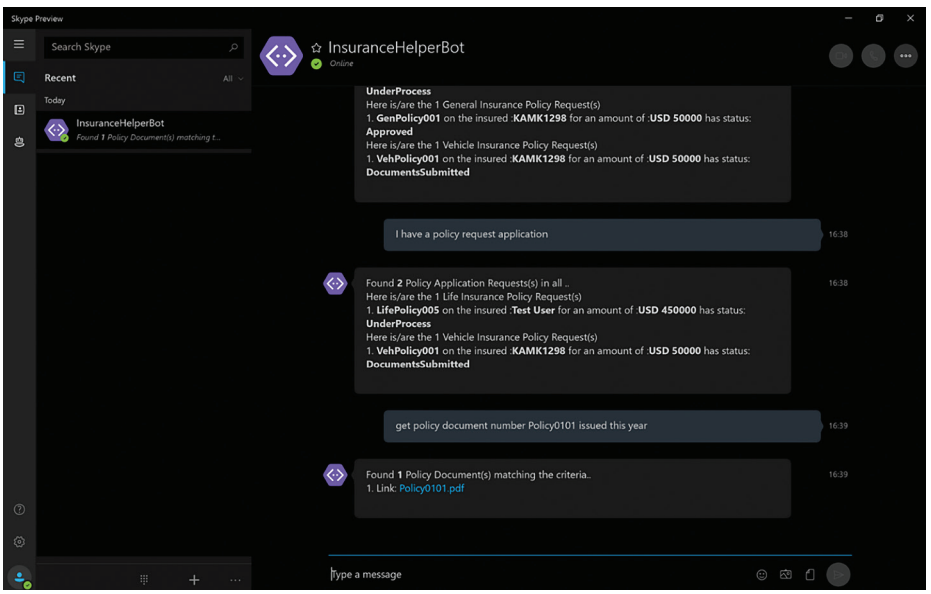


Figure 11 **Accessing the Bot from Skype**

Enabling the Bot Across Channels

In the Bot registry at dev.botframework.com/bots, you can enable the bot on other channels. **Figure 10** shows the bot application enabled across the Skype and Slack channels. Enabling the bot application for Slack requires some additional steps. A wizard is launched when you choose to add the bot to this channel, which takes you through the appropriate steps.

Figure 11 shows the bot application when accessed from Skype.

Wrapping Up

The Microsoft Bot Framework and other supporting technologies like LUIS and Azure Search provide the right foundation to build intuitive bot applications that let customers interact with your line-of-business applications. The way each of these services fit into the overall architecture of the solution ensures that the individual components can be matured or improved independent of the others. For example, the LUIS model can be tweaked to support more utterances that users could use to interact with the bot, while entailing no changes on the core bot application or the content in Azure Search. Once deployed, a bot application can, without any additional code changes, be enabled for access from various channels, and more channels can be added over time, to increase the reach and use of the bot application. ■

Srikantan Sankaran *is a technical evangelist from the DX team in India, based out of Bangalore. He works with numerous ISVs in India and helps them architect and deploy their solutions on Microsoft Azure. Reach him at sansri@microsoft.com.*

Thanks to the following Microsoft technical experts for reviewing this article: *Sandeep Alur and Hemanth Kathuria*

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

AUSTIN, TX
MAY 15-18, 2017
HYATT REGENCY

EVENT PARTNER
Microsoft

SUPPORTED BY
Visual Studio
msdn magazine
Visual Studio MAGAZINE

PRODUCED BY
1105 MEDIA

Capture and Analyze Brain Waves with Azure IoT Hub, Part 2

Benjamin Perkins

In Part 1 of this article (msdn.com/magazine/mt788621), there's a section where I discuss the mapping of human characteristics to object-oriented programming techniques such as inheritance and polymorphism. For example, inheritance of traits and behaviors from parent entities—such as eye color—can be directly related to a parent, or they can be derived from the root Human class where a derivative of arms, legs and hands are attained. That's also the case for polymorphic methods, such as speech, where a child might be able to converse in multiple languages, while the parent is able to do so in only one language. That's followed by a realization that although the ability to map human characteristics is completely possible, once the class is instantiated, it can't do anything without a brain. Without a brain, the child class is a motionless, thoughtless entity. This realization leads to the aspiration of three goals:

1. The improvement of machine decision-making logic based on definable brain patterns.
2. The enhancement of our own cognitive speed and accuracy.
3. The ability to control devices with brain waves based on pattern match.

This article discusses:

- Capturing, storing, and analyzing brain waves
- Creating a SQL Azure Database
- Creating a Stream Analytics Interface
- Using Power BI to analyze brain waves

Technologies discussed:

SQL Azure, Stream Analytics, Power BI

First, the brain is capable of processing data from numerous inputs—for example, sight, sound, smell, taste and touch—and then use those signals, along with any cognitive analysis, to trigger a reaction. If we can better understand how the brain accomplishes this processing, then we might find better coding techniques than the traditional if/then/else, try/catch, case/switch or recursion techniques, to name a few.

Second, after getting a better understanding of ourselves through understanding the brain, enhancing our cognitive abilities with artificial intelligence (AI) can help improve precision and speed of thought.

Third, using a brain computer interface (BCI) to capture and convert brain waves into physical or virtual data measurements allows for the storage and analysis of the brain waves. Once these brain wave patterns are reproducible, the ability to control objects such as cars, or take actions such as turning on lights, can be performed just by thought.

In Part 1 of this article I discussed the numerous technologies to capture, store, and analyze brain waves, each of which are briefly described in **Figure 1**. In Part 1 I discussed the configuration of the BCI and the creation of the Azure IoT Hub and the insertion of the brain waves into it.

Part 2 of this article provides detailed discussion of the remaining three components: the creation of the SQL Azure database to store the brain wave frequency values, Stream Analytics that move the data between the Azure IoT Hub and the SQL Azure database, and analysis of the data using Power BI.

Figure 1 **Components of the Brain Analyzation Project**

Components	Role	Brief Description
Emotiv Insights SDK	Capture	A brain interface that converts brain waves to numbers
Azure IoT Hub	Storage	Temporary storage queue for IoT device-rendered data
SQL Azure	Storage	Highly scalable, affordable and elastic database
Stream Analytics	Storage	An interface between Azure IoT and SQL Azure
Power BI	Analysis	Data analysis tool with simple graphic based support

Create the SQL Azure Instance and Data Table

Creating a SQL Azure database is straightforward: Within the Azure Portal select +New, then the Data + Storage menu item and, finally, SQL Database. This results in the rendering of a blade requesting required information to create the database like database name, server name, user id, and password. Note this information as it's needed later when the Stream Analytics job gets created.

To connect to the database, there are numerous options. I use the SQL Server Object Explorer in Visual Studio, as shown in **Figure 2**. It's also possible to use the SQL Database Management Console accessible from within a Web browser using the name of your database server followed by <servername>.database.windows.net. Note: A firewall rule is required to give the IP address of your client access to the database. If you get an error trying to connect, make sure you've selected +Add client IP on the firewall settings of the newly created database.

Once the database is created and accessible, create a database table named Measurement by right-clicking on the Tables folder as shown in **Figure 2**. The database table structure that works in this example is illustrated in **Figure 3** and is contained in the BrainComputerInterface.sql file of the downloadable sample code.

The database table is designed to support many types of BCIs with any number of electrodes/contacts and frequencies. The full database structure isn't provided, as it would become too complicated to implement. Rest assured that the Measurement database table is enough to capture the brain waves for analysis. **Figure 4** describes the columns in more detail.

The full database structure contains database tables for each of the columns containing an *Id in the name. Those *Id columns actually represent a Foreign Key to the primary tables containing the specific details about the *Id stored in the Measurement database table. For example, the ActivityId value stored in this database table would be 1, 2, 3, 4, 5, and so on. Those values would then have a linked description in a database table named Activity, where, 1=Smell a Flower, 2=In the Sun, 3=Firecracker and so on.

If required, by using a "Join," instead of rendering the numeric value on the Measurement database table, the description can be displayed. This is also one of the reasons why the primary key of the Measurement database table contains all the *Id columns; it's because the columns containing an *Id link back to another table containing the description of the value.

And last, creating a highly efficient data storage solution is a sophisticated and somewhat complicated venture. If you want the best possible solution, it's recommended to consult an expert in the field.

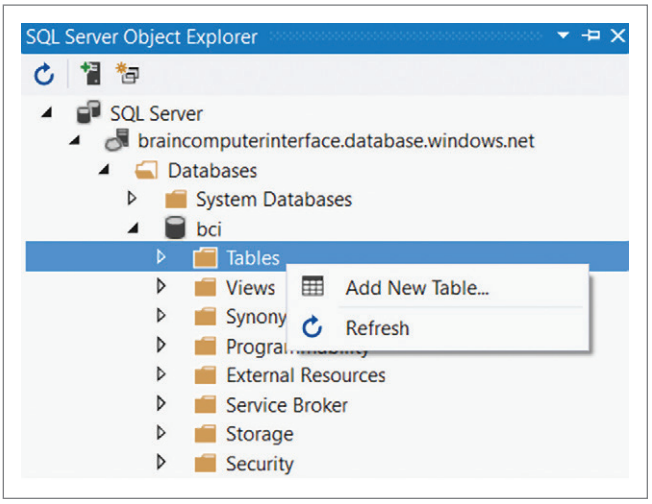


Figure 2 **Access a SQL Azure Database Using Visual Studio**

> The database table is designed to support many different types of BCIs with any number of electrodes/contacts and frequencies.

Create the Stream Analytics interface

As discussed in Part 1, getting the data stored in the Azure IoT Hub doesn't result in the data being stored in a location where analysis can happen. The data sent to the Azure IoT Hub must have a program or process that monitors the state of the entries and takes an action. An example of a program that can monitor and output data stored in the Azure IoT Hub can be found at bit.ly/2dfJJEo. But in this example, a Stream Analytics job is instead created.

To create a Stream Analytics job to monitor the Azure IoT Hub and move the data to the SQL Azure database created in the last section, the following actions are required:

- Create a Stream Analytics job
- Add the Input job

Figure 3 **Measurement Database Table**

```
CREATE TABLE [dbo].[MEASUREMENT] (
  [ManufacturerId]   INT       NOT NULL,
  [HardwareId]     INT       NOT NULL,
  [ActivityId]     INT       NOT NULL,
  [ChannelId]      INT       NOT NULL,
  [DeviceId]       INT       NOT NULL,
  [MeasurementId]    INT       IDENTITY (1, 1) NOT NULL,
  [UserName]       NVARCHAR (50)  NOT NULL,
  [GlobalDateTime]   DATETIME    DEFAULT (getdate()) NULL,
  [MeasurementDateTime] DATETIME    NULL,
  [THETA]        NUMERIC (18, 8) NULL,
  [ALPHA]        NUMERIC (18, 8) NULL,
  [LOWBETA]       NUMERIC (18, 8) NULL,
  [HIGHBETA]      NUMERIC (18, 8) NULL,
  [GAMMA]        NUMERIC (18, 8) NULL,
  CONSTRAINT [PK_MEASUREMENT]
  PRIMARY KEY ([ManufacturerId], [HardwareId], [ActivityId],
    [ChannelId], [DeviceId], [MeasurementId], [UserName]));
```

- Add the Output job
- Create the query to run on the incoming data

Once these steps are completed, the brain waves will be ready for analysis.

Create a Stream Analytics Job

To begin, access the Azure Portal and select + New, then Internet of Things and, finally, Stream Analytics job. Enter the Job name and other required details such as Subscription, Resource group, Location and click the Create button.

Add the Input Job

The input configuration is the place where the Stream Analytics job retrieves the information about the location to monitor incoming data. To create one, from within the Job Topology tile on the just-created Stream Analytics job, click on the Inputs journey and click the + Add link on the newly rendered blade to create a new input. Enter the name of the Input, for example, "FromIoTHub," then choose IoT hub from the Source dropdown, select service from the Shared access policy name dropdown, and leave the other settings with their default values. Click the Create button.

Notice that there's an option to change the Service access policy key from iothubowner to service. As discussed in Part 1 where the device identity is created, iothubowner has the permission to create new devices, letting them insert data into the IoT hub, while the service policy has only the ability to send and receive on the cloud-side endpoints, which is likely enough for what is being performed here. As there's no need to create new device identities to monitor and act on the received data, it's recommended to use the service policy; only use iothubowner if there's a reason to do so. Also, observe that the Service access policy key is automatically retrieved based on the IoT Hub selected from the IoT Hub dropdown; this is very helpful and intuitive.

Add the Output Job

The output configuration is the place where the Stream Analytics job retrieves the information about where to send the incoming data. To create one from within the Job Topology tile on the just-created Stream Analytics job, click on the Outputs journey, and click the + Add link on the newly rendered blade to create a new output. Enter the name of the Output—for example, ToSQLAzure—then choose SQL Database from the Sink dropdown, which will modify the blade, letting you select the databases in the given subscription.

It's possible to send the data to a SQL Azure database not in the same subscription by selecting the Provide SQL database setting manually from the Subscription dropdown. If you choose this option, provide the required information and then create the output job by clicking the Create button, which validates the manually entered information.

The output configuration is the place where the Stream Analytics job retrieves the information about where to send the incoming data.

If the database, which is to receive the data from the Azure IoT Hub, is in the same subscription, then select the database from the Database dropdown and enter the Username, Password and Table (example: Measurement). Click the Create button and after the Testing Output is successful, confirmed by the message, "connection to output 'ToSQLAzure' succeeded," the output job creation is complete.

Create the Query to Run on Incoming Data

The query created for this solution is a simple one. The query simply takes all the columns of a row from the input and inserts them into the configured output SQL Azure database. It's possible to run more complex queries that perform a kind of real-time analysis of the data that can alert someone or send the data to a different endpoint based on the outcome of the query. That's truly one of the great advantages available on the Azure platform in regard to IoT solutions: the real-time analysis of data coming from a large number of devices and the ability to immediately react once a recognized pattern is observed or a threshold breached.

Create and save the query in the current portal found at bit.ly/2bA4vAn and as illustrated in **Figure 5**. Clicking on the Query journey strategically and purposefully placed between the Inputs and Outputs journeys opens a new blade that supports the creation, saving, and testing of the query.

As shown in **Figure 6**, the Query matches the attributes of the brain Activity class contained within the SendBrainMeasurementToAzureAsync method in the sample BrainComputerInterface. The SendBrainMeasurementToAzureAsync method is the place where the brain wave frequency measurements for a given electrode are sent to the Azure IoT Hub. Start the Stream Analytics job to begin the monitoring of the Azure IoT Hub. The Query is used to perform the retrieval and removal of the measurement from the Azure IoT Hub and used again as the basis for the insertion of the row into the Measurement database table.

To test the query, you need to use the auxiliary portal at bit.ly/1tPjlg7. Navigate to the auxiliary portal and select the Test button, which opens a

Figure 4 **Column Descriptions in Measurement Database Table**

Column Name	Description
ManufacturerId	The company that manufactured the device (ex: Emotiv)
HardwareId	The specific device type (ex: Insight vs. EPOC+)
ActivityId	The scenario or session (ex: smelling a flower)
ChannelId	The contact or electrode (ex: AF3, AF4, Tz, etc.)
DeviceId	A device id bound to a specific user
MeasurementId	The unique identifier for the row in the table
UserName	The user name used to create the device identity
Global/MeasurementDateTime	Time of insertion from the client and on the server
THETA, ALPHA, LOWBETA, etc.	The brain wave frequency reading for a specific electrode

Spreadsheets Made Easy.

Fastest Calculations

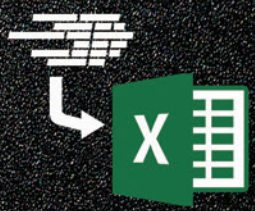
Evaluate complex Excel-based models and business rules with the fastest and most complete Excel-compatible calculation engine available.

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

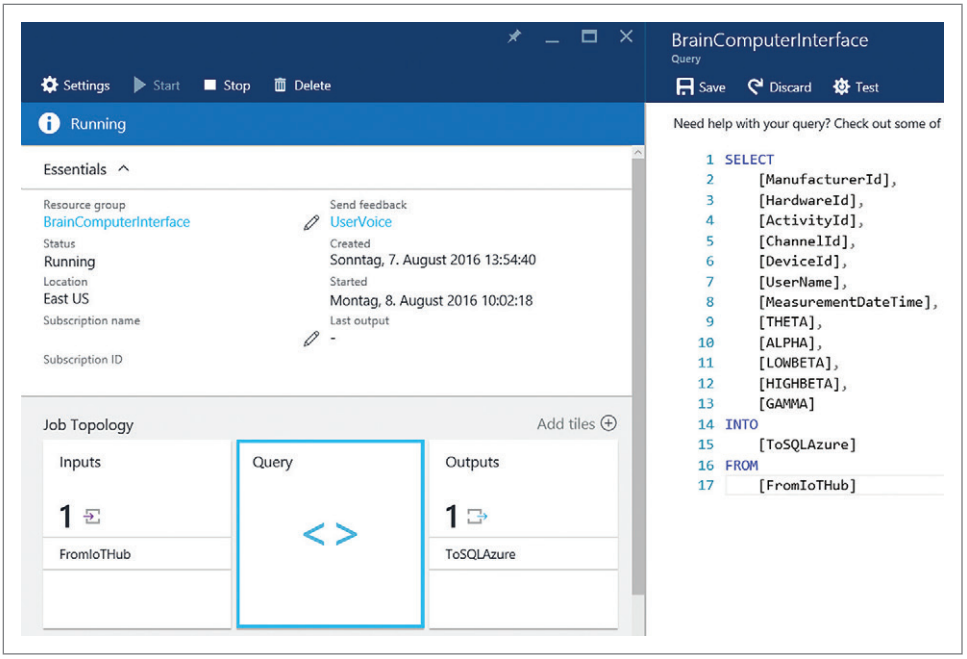


Figure 5 **Creating the Stream Analytics Query**

dialog box requesting a sample input file. There's a file named MEASUREMENTData.json in the downloadable code that can be used here. Run the test and confirm there were no exceptions.

> The most important aspect of capturing the data is that it happens within a scenario that can be reproduced.

That concludes the creation of the components for uploading brain waves to the cloud. It was quite a journey that required the implementation of four entities: the code to convert the electrode frequency readings into a number, the construction of the Azure IoT Hub that accepts those numbers, a SQL Azure database to permanently store those numbers, and a Stream Analytics job to manage the state of the numbers and move them to the permanent data storage. Now you can start to upload the brain waves to the cloud and store them in a database for analysis. Analyzing the data is discussed in detail in the next section, but take a moment to pat yourself on the back if you've successfully made it this far.

Analyzing the Brain Waves

Before discussing the actual topic for this section, it's important to first mention the process that was followed to acquire the data. I touched on many of the topics in a recent blog post (bit.ly/29LbKEe), however, it's worthwhile to mention them specifically.

The most important aspect of capturing the data is that it happens within a scenario that can be reproduced. For example, if you wanted to capture brain waves while reading a book, listening to music or watching a movie, it's important that each time you do, you read the same part of the same book, listen to the same song, and watch the same part of a movie. I have no scientific proof of this other than a few experiments I ran on myself, but when I captured my brain waves while listening to music, the pattern was different depending on the kind of music I was listening to. As well, although I watched the same movie, I watched different parts, which resulted in different patterns. You would agree that movies sometimes evoke different emotions and those would trigger a different pattern and measurements of the brain activity. In contrast, when I captured the measurements while in a meditative state, where in all sessions the environment was quiet and dark, there was indeed a close pattern in my brain activity, or lack thereof, in the data.

Not only is it critical to have the same scenarios replicated between sessions, it's also important to make sure the device capturing the brain waves is functioning the same in all cases. I touched on this in Part 1 if you recall. This is why there's code to check the signal strength and battery level prior to beginning the recording of brain waves, as well as the recommendation to use the online tool, which provides a graphical representation of the electrodes\

Figure 6 **Inserting the Brain Wave into Azure IoT Hub**

```
while (true)
{
for (int i = 0; i < 5; i++)
{
  engine.IEE_GetAverageBandPowers(0, channelList[i],
    theta, alpha, low_beta, high_beta, gamma);
  SendBrainMeasurementToAzureAsync(channelList[i].ToString(), theta[0].ToString(),
    alpha[0].ToString(), low_beta[0].ToString(),
    high_beta[0].ToString(), gamma[0].ToString());
  }
}

private static async void SendBrainMeasurementToAzureAsync(string channel,
  string theta, string alpha, string lowbeta, string highbeta,
  string gamma)
{
  // ...
  try
  {
   var brainActivity = new
    { ManufacturerId, HardwareId, ActivityId, ChannelId,
      DeviceId, UserName, MeasurementDateTime, theta,
      alpha, lowbeta, highbeta, gamma };
  var messageString = JsonConvert.SerializeObject(brainActivity);
  var message = new Message(Encoding.ASCII.GetBytes(messageString));
  await deviceClient.SendEventAsync(message);
catch (Exception ex)
{ // ...}
}
```

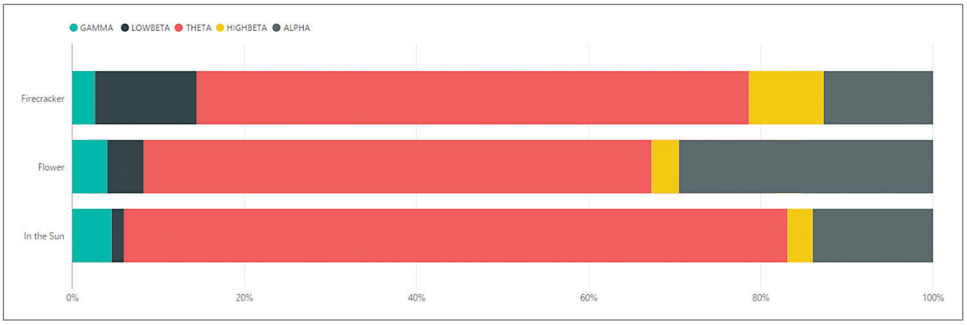


Figure 7 **Analysis of Brain Waves Using Power BI**

contact values. To get valid comparisons, it's imperative that the environment, actions, and device are as identical as possible between sessions; otherwise, it's not certain what actually triggered the brain activity and comparing two unlike sessions adds no value and results in inconclusive assumptions. Now that that's all clear, it's time for the fun part.

For this example, the Power BI Desktop version is used and is a free download currently available at bit.ly/1S8XkLO. Once installed and running, while on the Home tab, click the Get Data menu item, which shows the numerous sources to which Power BI can connect. Select SQL Server and enter the server name where the database is running (the one that was created in the "Create the SQL Azure Instance and Data Table" section) and enter it into the configuration wizard along with the optional database name. Note that the server name is not the database name, rather server name is the name of the server on which the database is running, which can be found in the Azure Portal on the Properties blade of the SQL Azure server. It resembles something like this: <servername>.database.windows.net. Once entered, click the OK button and enter the credentials for the database, then click Connect. Don't forget to add your client IP address so the firewall lets the connection pass through.

> This scalable scenario can support millions of measurements per second, while at the same time it's also feasible for an individual hobbyist, like me.

After successful connection, continue with the configuration wizard and select the Measurement table on the Navigator window, then click the Load button. The data is now ready to be analyzed and compared. An example of a "100% Stacked bar chart" where Axis is ActivityId and Values are GAMMA, LOWBETA, THETA, HIGHBETA and ALPHA, is illustrated in **Figure 7**.

Using the meanings of each frequency (GAMMA, LOWBETA, THETA, HIGHBETA and ALPHA) you can draw some conclusions and determine for yourself if those conclusions make sense. Take, for example, the brain waves collected while smelling a flower. Notice in **Figure 7** that the ALPHA measurement consumes a large percentage of that session. When I recall my state of mind when I smelled the flower, it was a relaxing and reflective feeling and that matches what's illustrated in **Figure 7**. Even more apparent is when I was startled by a loud noise, what I called Firecracker, resulted in some higher HIGHBETA reading. HIGHBETA is commonly associated with Focus and quick thinking, which is a match for what would be expected in a situation like that.

The next step is to capture those same sessions again, in as much of an identical environment as possible, and compare the results. If, over time, an identifiable pattern can be determined that happens only during a given scenario, then taking an action to control something physical using the brain is only a simple if statement away. Just imagine having your BCI connected to a device, you smell a flower and Cortana says, "Do you like how that flower smells? Describe it or take a picture and I'll tell you what kind it is." This can be accomplished using the Computer Vision API, which is part of the Microsoft Cognitive Services offerings.

Wrapping Up

In this article, you learned how to set up the four pieces required to load brain waves from a BCI into the cloud. This scalable scenario can support millions of measurements per second, while at the same time it's also feasible for an individual hobbyist, like me. The fact is, the more data we share in a collective sense the more we can learn about ourselves, not only as individuals but also as interconnected beings. Progressing from an early attempt to create virtual life using C# classes and object-oriented programming techniques, I've come to the point where giving the Human class the actual means to respond and react to real-world events is an actual possibility. Although we aren't at that point now, there are only a few more steps to a path toward an intelligent virtual entity that really changes everything; we're on the brink. One path to that end has been set through the ability to capture and analyze brain waves in given scenarios, analyzing those patterns, translating them to code and introducing an interface to them. Many of these APIs are already available as part of the Microsoft Cognitive Services offerings.

But I believe the greatness of this comes not from brain waves and patterns of a single individual, rather they come from patterns found from the greater population of humans. Finding the small things that bring us together, the subtle elements that we all share, will lead us to great places. ■

Benjamin Perkins *is an escalation engineer at Microsoft and author of four books on C#, IIS, NHibernate and Microsoft Azure. He recently completed coauthoring "Beginning C# 6 Programming with Visual Studio 2015" (Wrox). Reach him at benperk@microsoft.com.*

Thanks to the following Microsoft technical expert for reviewing this article: *Sebastian Dau*

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

LAS VEGAS
MAR 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
YOUR GROWTH. OUR BUSINESS.

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

TURN THE PAGE FOR FULL AGENDA DETAILS

Sunday Pre-Con Hands-On Labs

Choose From:

- Angular
- Azure
- XAML

NEW! ONLY $595

SPACE IS LIMITED

Register by December 16 and Save $500!*

Use promo code VSLDEC4 Scan the QR code to register or for more event details. *SAVINGS BASED ON 5 DAY PACKAGES ONLY.

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!

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

AGENDA AT-A-GLANCE

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

CONNECT WITH VISUAL STUDIO LIVE!

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

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

ALM / DevOps | **Cloud Computing** | **Database and Analytics** | **Mobile Client** | **Software Practices**

START TIME	END TIME	
Full Day Hands-On Labs: Sunday, March 12, 2017 *(Separate entry fee required)*		
7:30 AM	9:00 AM	Pre-Conference Workshop Registration - Coffee and Morning Pastries
9:00 AM	6:00 PM	**HOL01** Full Day Hands-On Lab: Build an Azure App in a Day - *Brian Randell*
1:00 PM	2:00 PM	Lunch @ Le Village Buffet, Paris Las Vegas
2:00 PM	6:00 PM	Workshop Continues
Pre-Conference Workshops: Monday, March 13, 2017 *(Separate entry fee required)*		
7:30 AM	9:00 AM	Pre-Conference Workshop Registration - Coffee and Morning Pastries
9:00 AM	6:00 PM	**M01** Workshop: Native Mobile App Development for iOS, Android and Windows Using C# - *Marcel de Vries & Roy Cornelissen*
1:00 PM	2:00 PM	Lunch @ Le Village Buffet, Paris Las Vegas
2:00 PM	6:00 PM	Workshop Continues
7:00 PM	9:00 PM	Dine-A-Round
Day 1: Tuesday, March 14, 2017		
7:00 AM	8:00 AM	Registration - Coffee and Morning Pastries
8:00 AM	9:00 AM	Keynote: To Be Announced Donovan Brown, Senior DevOps Program Manager, US Developer Division Team,
9:15 AM	10:30 AM	**T01** Essential Web Development with ASP.NET Core - *Mark Michaelis* / **T02** An Overview of the Xamarin Programming Platforms - *Laurent Bugnion*
10:45 AM	12:00 PM	**T06** Migrating to ASP.NET Core - A True Story - *Adam Tuliper* / **T07** Building Truly Universal Applications with Windows, Xamarin and MVVM - *Laurent Bugnion*
12:00 PM	1:00 PM	Lunch
1:00 PM	1:30 PM	Dessert Break - Visit Exhibitors
1:30 PM	2:45 PM	**T11** An Introduction to TypeScript - *Jason Bock* / **T12** What's New for Developers in SQL Server 2016 - *Leonard Lobel*
3:00 PM	4:15 PM	**T16** Assembling the Web - A Tour of WebAssembly - *Jason Bock* / **T17** No Schema, No Problem! Introduction to Azure DocumentDB - *Leonard Lobel*
4:15 PM	5:30 PM	Welcome Reception
Day 2: Wednesday, March 15, 2017		
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries
8:00 AM	9:15 AM	**W01** Angular 101: Part 1 - *Deborah Kurata* / **W02** Customizing Your UI for Mobile Devices: Techniques to Create a Great User Experience - *Laurent Bugnion*
9:30 AM	10:45 AM	**W06** Angular 101: Part 2 - *Deborah Kurata* / **W07** Busy .NET Developer's Guide to Native iOS - *Ted Neward*
11:00 AM	12:00 AM	General Session: To Be Announced
12:00 PM	1:00 PM	Birds-of-a-Feather Lunch
1:00 PM	1:30 PM	Dessert Break - Visit Exhibitors - Exhibitor Raffle @ 1:15pm (Must be present to win)
1:30 PM	2:45 PM	**W11** User Authentication for ASP.NET Core MVC applications - *Brock Allen* / **W12** Cloud Enable an Existing WPF LOB App - *Robert Green*
3:00 PM	4:15 PM	**W16** Securing Web APIs in ASP.NET Core - *Brock Allen* / **W17** Strike Up a Conversation with Cortana on Windows 10 - *Walt Ritscher*
4:30 PM	5:45 PM	**W21** ASP.NET Core 1.0 Tag Helpers - *Robert Boedigheimer* / **W22** Busy Developer's Guide to NoSQL - *Ted Neward*
7:00 PM	8:30 PM	Experience The LINQ Vortex & High Roller Event
Day 3: Thursday, March 16, 2017		
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries
8:00 AM	9:15 AM	**TH01** Debugging Your Website with Fiddler and Chrome Developer Tools - *Robert Boedigheimer* / **TH02** Windows for Makers: Raspberry Pi, Arduino & IoT - *Nick Landry*
9:30 AM	10:45 AM	**TH06** I Say A "Front-end Build Pipeline", You Say WAT!? - *Chris Klug* / **TH07** Building Connected and Disconnected Mobile Apps - *James Montemagno*
11:00 AM	12:15 PM	**TH11** JavaScript Patterns for the C# Developer - *Ben Hoelting* / **TH12** Building Cross-Platform Business Apps with CSLA .NET - *Rockford Lhotka*
12:15 PM	1:45 PM	Lunch
1:45 PM	3:00 PM	**TH16** Integrating AngularJS & ASP.NET MVC - *Miguel Castro* / **TH17** Native iOS and Android Development with C# and Xamarin - *James Montemagno*
3:15 PM	4:30 PM	**TH21** Increase Website Performance and Search with Lucene.Net Indexing - *Ben Hoelting* / **TH22** Building Cross-Platform C# Apps with a Shared UI Using Xamarin.Forms - *Nick Landry*
Post-Conference Workshops: Friday, March 17, 2017 *(Separate entry fee required)*		
7:30 AM	8:00 AM	Post-Conference Workshop Registration - Coffee and Morning Pastries
8:00 AM	5:00 PM	**F01** Workshop: Service Oriented Technologies - Designing, Developing, & Implementing WCF and the Web API - *Miguel Castro*

Speakers and sessions subject to change

Modern Apps LIVE!

MOBILE, CROSS-DEVICE & CLOUD DEVELOPMENT

Presented in partnership with Magenic

BONUS CONTENT! Modern Apps Live! is now a part of Visual Studio Live! Las Vegas at no additional cost!

Visual Studio / .NET Framework	Web Client	Web Server	Windows Client	Modern Apps Live!
Full Day Hands-On Labs: Sunday, March 12, 2017 *(Separate entry fee required)*				
Pre-Conference Workshop Registration - Coffee and Morning Pastries				
	HOL02 Full Day Hands-On Lab: AngularJS 2 - *Ted Neward*	**HOL03** Full Day Hands-On Lab: An Introduction to Building XAML Applications - *Billy Hollis*		**Modern Apps Live!**
Lunch @ Le Village Buffet, Paris Las Vegas				
	Workshop Continues	Workshop Continues		
Pre-Conference Workshops: Monday, March 13, 2017 *(Separate entry fee required)*				
Pre-Conference Workshop Registration - Coffee and Morning Pastries				
	M02 Workshop: Developer Dive into SQL Server 2016 - *Leonard Lobel*	**M03** Workshop: Distributed Cross-Platform Application Architecture - *Rockford Lhotka & Jason Bock*		**M04** Workshop: Building Modern Mobile Apps - *Brent Edwards & Kevin Ford*
Lunch @ Le Village Buffet, Paris Las Vegas				
	Workshop Continues	Workshop Continues		Workshop Continues
Dine-A-Round				
Day 1: Tuesday, March 14, 2017				
Registration - Coffee and Morning Pastries				
Microsoft				
	T03 Developer Productivity in Visual Studio "15" - *Robert Green*	**T04** Understanding the VR/AR Landscape - *Katherine Harris*		**T05** Modern App Development: Transform How You Build Web and Mobile Software - *Rockford Lhotka*
	T08 Roll Your Own Dashboard in XAML - *Billy Hollis*	**T08** DevOps and Azure with the MS Stack - *Abel Wang*		**T10** Architecture: The Key to Modern App Success - *Brent Edwards*
Lunch				
Dessert Break - Visit Exhibitors				
	T13 A Developers Introduction to HoloLens - *Billy Hollis & Brian Randell*	**T14** To Be Announced		**T15** Manage Distributed Teams with Visual Studio Team Services and Git - *Brian Randell*
	T18 Essential C# 7.0 - *Mark Michaelis*	**T19** To Be Announced		**T20** Focus on the User Experience #FTW - *Anthony Handley*
Welcome Reception				
Day 2: Wednesday, March 15, 2017				
Registration - Coffee and Morning Pastries				
	W03 What's New in Azure IaaS v2 - *Eric D. Boyd*	**W04** Application Lifecycle Management (ALM) - *Brian Randell*		**W05** DevOps, Continuous Integration, the Cloud, and Docker - *Dan Nordquist*
	W08 Microservices with Azure Container Service & Service Fabric - *Vishwas Lele*	**W09** Team Foundation Server 2015: Must-Have Tools and Widgets - *Richard Hundhausen*		**W10** Mobile Panel - *James Montemagno, Ryan J. Salva, Kevin Ford, Rockford Lhotka*
General Session: To Be Announced				
Birds-of-a-Feather Lunch				
Dessert Break - Visit Exhibitors - Exhibitor Raffle @ 1:15pm (Must be present to win)				
	W13 I'm Emotional - Using Microsoft Cognitive Services to Understand the World Around You - *Adam Tuliper*	**W14** Professional Scrum Development Using Visual Studio 2015 - *Richard Hundhausen*		**W15** C# Everywhere: How CSLA .NET Enables Amazing Cross-Platform Code Reuse - *Rockford Lhotka*
	W18 Cloud Oriented Programming - *Vishwas Lele*	**W19** Introduction to Containers and Docker - *Marcel de Vries*		**W20** Coding for Quality and Maintainability - *Jason Bock*
	W23 Practical Internet of Things for the Microsoft Developer - *Eric D. Boyd*	**W24** Using Docker on Windows in VSTS Build and Release Management - *Marcel de Vries*		**W25** Modern Mobile Development: Build a Single App For iOS & Android with Xamarin Forms - *Kevin Ford*
Experience The LINQ Vortex & High Roller Event				
Day 3: Thursday, March 16, 2017				
Registration - Coffee and Morning Pastries				
	TH03 Accelerate Your Mobile App Development with Azure App Services Mobile Apps - *Brian Noyes*	**TH04** Agile: You Keep Using That Word - *Philip Japikse*		**TH05** Modern Web Development: ASP.NET MVC and Web API - *Allen Conway*
	H08 Connect All The Things with Azure Service Bus, Notification Hubs, Event Hubs, and IoT Hubs - *Brain Noyes*	**TH09** Visualizing the Backlog with User Story Mapping - *Philip Japikse*		**TH10** Modern Web Development: Building a Smart Web Client with TypeScript and Angular2 - *Allen Conway*
	TH13 Add A Conversational User Interface to Your App with the Microsoft Bot Framework - *Walt Ritscher*	**TH14** End-to-End Dependency Injection & Testable Code - *Miguel Castro*		**TH15** Cloud Panel - *Rockford Lhotka*
Lunch				
	TH18 Introduction to R and Microsoft R Server - *James McCaffrey*	**TH19** Open Source Software for Microsoft Developers - *Rockford Lhotka*		**TH20** Universal Windows Development: UWP for PC, Tablet & Phone - *Nick Landry*
	TH23 Introduction to Azure Machine Learning - *James McCaffrey*	**TH24** SOLID - The Five Commandments of Good Software - *Chris Klug*		**TH25** Using All That Data: Power BI to the Rescue - *Scott Diehl*
Post-Conference Workshops: Friday, March 17, 2017 *(Separate entry fee required)*				
		F02 Workshop: Application Lifecycle Management (ALM) - *Brian Randell*		**F03** Workshop: Modern App Deep Dive: Xamarin, Responsive Web, UWP, CSLA .NET - *Kevin Ford, Jason Bock, Brent Edwards, Allen Conwa*

vslive.com/lasvegas

DAVID S. PLATT

What, Me Mentor?

I'm in real trouble now. I signed up to mentor Annabelle's high school robotics team. More than that, I agreed to take charge of the emerging software group. Our kickoff meeting is the day after tomorrow. I am so scrod (a Boston term, look it up).

We're team 5459, the Ipswich Tigers (ipswich5459.com, new sponsors always welcome). We belong to the FIRST Robotics league (bit.ly/2dCyhoP), founded by Dean Kamen (bit.ly/2e301xq), inventor of the Segway scooter. Every January, the league announces new robotic tasks for the coming season. The teams then have six weeks to design and build a robot that accomplishes these tasks, with which they compete in live meets starting in March.

Annabelle joined the team in its first year, when she was in eighth grade. She was searching for her tribe, as eighth graders do. She found it in the camaraderie of the robotics lab, where brainpower and creativity are admired and respected, especially when coupled with hard work. The next year, as a high school freshman, she served as head of the mechanical division. This year she'll repeat that role, while training an understudy. I can envision her as team captain her senior year.

Software somehow didn't catch her fancy. (Where have I gone wrong?) She says that Christine, a mechanical engineer and the team's founding mentor, is the smartest person she knows. "Present company excluded, of course?" I asked her. "Nope. Deal with it." Oy. Acorns and trees, I guess.

The students bumbled through the first two years, mostly on their own, chopping and hacking without really thinking things through. As you'd expect, they didn't place highly, but they sure did have fun. They learned what not to do by doing it wrong and falling on their faces. This year, they're determined to learn from their previous mistakes. They approached their mentors at the end of the summer, asking us to help them raise their game. "We know we need your help, but we don't know what help we need," Annabelle said. So we're stepping up, like the fools we are.

Figure 1 **Team 5459, the Ipswitch Tigers (center), competing in the FIRST Robotics league.**

My role is to guide their software group: advising, cajoling, wheedling, occasionally threatening them into some semblance of a modern software organization. The timetable is so tight that we'll have to use Agile techniques. I'll start by insisting that students show up on time for meetings and turn off their damn smartphones, and we'll go from there.

My mentee students all use the buzzword "coding" to describe their activity. In my first rant to them, I explained that coding is only one small part of software development. More important, I argued, is defining the computing problem, deciding what needs to be coded. Then you have to figure out which people are going to do which parts of it, how they will communicate with each other, how the pieces fit together, and how you're going to verify it all. I have trouble getting this idea through the heads of college students and working developers. Maybe these younger kids will be more teachable.

Here's what inspires me: These kids don't give up. Ever. At last spring's competition, they kept banging, banging, banging; modifying their robot on the fly (**Figure 1**); despite blood, sweat, tears and soldering iron burns; working their butts off to improve from 27th place overall to 25th place in the very last bout. And then celebrating it, as the accomplishment that it truly was. Lump in the throat time, my friends.

What a privilege and honor to help shape this team and its members. And what a responsibility, too—to all the kids, their families, my profession, my world; and yours, too, dear reader. It sits more heavily than I thought it would.

I wrote the team a theme song, which they deemed too old-farty to sing. ("Bots to Build," to the tune of "Boats to Build," with apologies to Guy Clark, see bit.ly/2enSss8.) The verses are, mercifully, forgotten. But I can't help humming the chorus under my breath, with which I will leave you as Team 5459 sets sail for whatever comes:

We're gonna build us a bot, with all our hands
5459's got a plan
Let the wheels roll where they will
'Cause we've got bots to build! ■

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

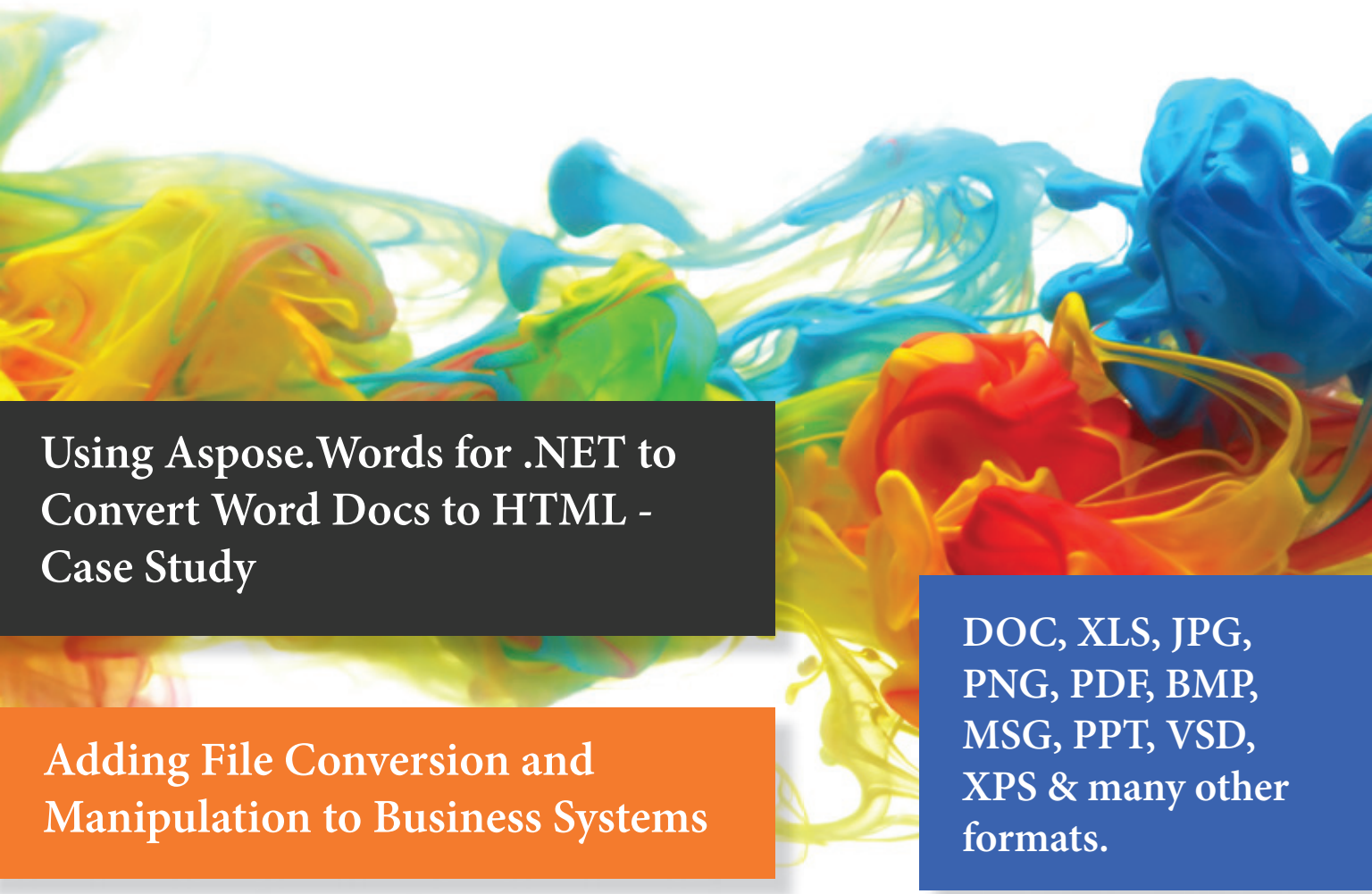
Visit us at www.groupdocs.com

ASPOSE

File Format APIs

Powerful File APIs that are easy and intuitive to use

Native APIs for.NET, Java & Cloud



Using Aspose.Words for .NET to Convert Word Docs to HTML - Case Study

Adding File Conversion and Manipulation to Business Systems

DOC, XLS, JPG, PNG, PDF, BMP, MSG, PPT, VSD, XPS & many other formats.

www.aspose.com

EU Sales: +44 141 628 8900

US Sales: +1 903 306 1676
sales@aspose.com

AU Sales: +61 2 8006 6987

Aspose.Cells

Work with spreadsheets and data without depending on Microsoft Excel

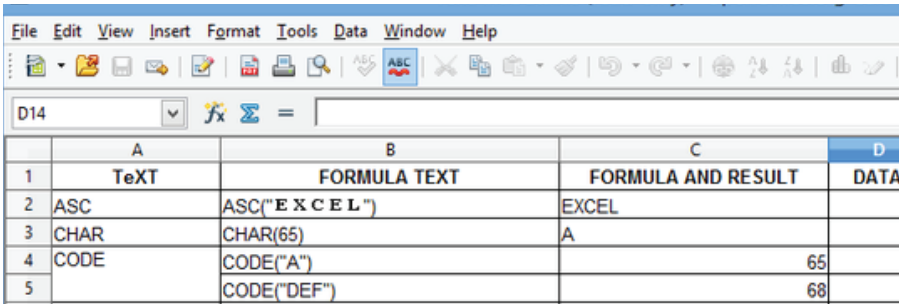
- Solution for spreadsheet creation, manipulation and conversion.
- Import and export data.

ASPOSE.CELLS IS A PROGRAMMING API that allows developers to create, manipulate and convert Microsoft Excel spreadsheet files from within their own applications. Its powerful features make it easy to convert worksheets and charts to graphics or save reports to PDF.

A flexible API for simple and complex spreadsheet programming.

Aspose.Cells speeds up working with Microsoft Excel files. The API is a flexible tool for simple tasks such as file conversion, as well as complex tasks like building models. Developers control page layout, formatting, charts and formulas. They can read and write spreadsheet files and save out to a wide variety of image and text file formats.

Fast, scalable, and reliable, Aspose.Cells saves time and effort compared to using Microsoft Office Automation.



	A	B	C	D
1	TeXT	FORMULA TEXT	FORMULA AND RESULT	DATA
2	ASC	ASC("EXCEL")	EXCEL	
3	CHAR	CHAR(65)	A	
4	CODE	CODE("A")	65	
5		CODE("DEF")	68	

Aspose.Cells lets developers work with data sources, formatting, even formulas.

Common Uses

- Building dynamic reports on the fly.
- Creating Excel dashboards with charts and pivot tables.
- Rendering and printing spreadsheets and graphics with high fidelity.
- Exporting data to, or importing from, Excel and other spreadsheets.
- Generating, manipulating and editing spreadsheets.
- Converting spreadsheets to images or other file formats.

Key Features

- A complete spreadsheet manipulation solution.
- Flexible data visualization and reporting.
- Powerful formula engine.
- Complete formatting control.

Supported File Formats

XLS, XLSX, XLSM, XMPS, XLTX, XLTM, ODS, XPS, SpreadsheetML, tab delim., CSV, TXT, PDF, HTML, and many image formats including SVG, TIFF, JPEG, PNG and GIF.

Format support varies across platforms.

Platforms

Pricing Info					
	Standard	Enhanced		Standard	Enhanced
Developer Small Business	$999	$1498	Site Small Business	$4995	$7490
Developer OEM	$2997	$4494	Site OEM	$13986	$20972

The pricing info above is for .NET: prices for other platforms may differ. For the latest, contact sales.

Aspose.Cells for

.NET, Java, Cloud & more

File Formats

XLS, CSV, ODS, PDF, SVG, HTML, PNG, BMP, XPS, JPG SpreadsheetML and many others.

Spreadsheet Manipulation

Aspose.Cells lets you create, import, and export spreadsheets and also allows you to manipulate contents, cell formatting, and file protection.

Creating Charts

Aspose.Cells comes with complete support for charting and supports all standard chart types. Also, you can convert charts to images.

Graphics Capabilities

Easily convert worksheets to images as well as adding images to worksheets at runtime.

100% Standalone

Aspose.Cells does not require Microsoft Office to be installed on the machine in order to work.

Get your FREE Trial at
http://www.aspose.com

ASPOSE
File Format APIs

Aspose.Words

Program with word processing documents independently of Microsoft Word

- Solution for document creation, manipulation and conversion.
- Advanced mail merge functionality.

ASPOSE.WORDS IS AN ADVANCED PROGRAMMING API that lets developers perform a wide range of document processing tasks with their own applications. Aspose.Words makes it possible to generate, modify, convert, render and print documents without Microsoft Word. It provides sophisticated and flexible access to, and control over, Microsoft Word files.

Generate, modify, convert, render and print documents without Microsoft Word.

Aspose. Words is powerful, user-friendly and feature rich. It saves developers time and effort compared to using Microsoft Office Automation and makes gives them powerful document management tools.

Aspose.Words makes creating, changing and converting DOC and other word processing file formats fast and easy.

	Table			
	Column 1	**Column 2**	**Column 3**	**Column 4**
Row 1	Cell 1	Cell 2	Cell 3	Cell 4
Row 2	Cell 1	Cell 2	Cell 3	
Row 3	Cell 1	Cell 2		

Aspose.Words has sophisticated controls for formatting and managing tables and other content.

Common Uses

- Generating reports with complex mail merging; mail merging images.
- Populating tables and documents with data from a database.
- Inserting formatted text, paragraphs, tables and images into Microsoft Word documents.
- Adding barcodes to documents.
- Inserting diagrams and watermarks into Word documents.
- Formatting date and numeric fields.

Key Features

- A complete Microsoft Word document manipulation solution.
- Extensive mail merge features.
- Complete formatting control.
- High-fidelity conversion, rendering and printing.

Supported File Formats

DOC, DOCX, ODT, OOXML, XML, HTML, XHTML, MHTML, EPUB, PDF, XPS, RTF, and a number of image formats, including TIFF, JPEG, PNG and GIF.

Format support varies across platforms.

Platforms

Pricing Info					
	Standard	Enhanced		Standard	Enhanced
Developer Small Business	$999	$1498	Site Small Business	$4995	$7490
Developer OEM	$2997	$4494	Site OEM	$13986	$20972

The pricing info above is for .NET: prices for other platforms may differ. For the latest, contact sales.

Case Study: Aspose.Words for .NET

ProHire Staffing - Using Aspose.Words for .NET to convert Word Docs to HTML

PROHIRE IS THE WORKFORCE SOLUTIONS LEADER IN THE UNITED STATES AND SPECIALIZE IN THE RECRUITMENT OF SALES AND SALES MANGEMENT PROFESSIONALS. We were founded with the goal of becoming the premier provider of executive search and placement services to the Fortune 500 and Inc. 500 Companies.

Problem

ProHire uses Bullhorn ATS as its Application Tracking System to track the electronic handling of its recruitment needs. We wanted to integrate the Bullhorn API with our new website. Our goal was to convert MS Word Documents resumes into a clean and concise HTML format into our existing .Net Stack. The converted HTML resume version needed to look close to the original.

> "The transitions with Aspose. Words for .NET was very painless to do."

Looking for a Solution

We chose the ASPOSE.Words product because it easily integrated into our existing .Net stack, and provided a quality MS Word to HTML conversion. The product was easy to download, and with a few lines of code we were up and running. We found the primary difference between the Aspose. Words and other products was the obvious conversion quality from MS Word to HTML.

Finding a Solution

We had tested other products that converted Word to HTML. Every one we tested had some problem with the conversion. Some of them lost elements of the resume during the conversion. Most of them changed the format of the resume or changed the color of the text unexpectedly. This is unacceptable when you are sending a resume to a hiring manger. We were very satisfied with the results. We did not need any technical support because documentation was sufficient to us.

Implementation

Once we had the Aspose DLL our developer was able to implement Aspose.Words for .NET in a few hours. The transitions with Aspose. Words for .NET was very painless to do.

Outcome

We are very pleased with the success of our Aspose.Words for .NET implementation. Aspose.Words is a very powerful development tool that is well documented and easy to install. The documentation is easy to understand and use. If you want a product to convert Word Docs to HTML look no further. ProHire is happy to recommend Aspose.

This is an extract from a case study on our website. For the full version, go to: www.aspose.com/corporate/customers/case-studies.aspx

The converted HTML resume version needed to look close to the original.

Open, Create, Convert, Print & Save Files

from within your *own* applications.

ASPOSE.TOTAL

allows you to process these file formats:

- Word documents
- Excel spreadsheets
- PowerPoint presentations
- PDF documents
- Project documents
- Visio documents
- Outlook emails
- OneNote documents

DOC XLS PPT PDF EML
PNG XML RTF HTML VSD
BMP & barcode images.

ASPOSE
File Format APIs

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

Helped over 11,000 companies and over 300,000 users work with documents in their applications.

File Format APIs

GET STARTED NOW

- Free Trial
- 30 Day Temp License
- Free Support
- Community Forums
- Live Chat
- Blogs
- Examples
- Video Demos

Adding File Conversion and Manipulation to Business Systems

How often do people in your organization complain that they can't get information in the file format and layout they want? Converting documents from one format to another without losing layout and formatting should be simple, but it can be frustrating for both users and developers.

EXTRACTING DATA FROM A DATABASE AND DELIVERING IT TO THE SALES TEAM AS A REPORT, complete with charts and corporate branding, is fine. Until the sales team says that they want it as a Microsoft Excel file, and could you add a dashboard?

Using information from online forms in letters that can are printed and posted is easy. But what if you also want to add tracking barcodes and archive a digital copy as a PDF?

> Aspose creates APIs that work independently of Microsoft Office Automation.

Ensuring that your business system supports all the different Microsoft Office file formats your users want can be difficult. Sometimes the native file format support of your system lets you down. When that is the case, use tools that extend that capability. A good tool can save you time and effort.

Document Conversion Options

Building your own solution: Time-consuming and costly, this option is only sensible if the solution you develop is central to your business.

Using Microsoft Office Automation: Microsoft Office Automation lets you use Microsoft Office programs server-side. It is not how the Office products were designed to be used. It can work well but you might notice issues with the stability, security and speed of the system, as well as cost.

Using an API: The API market has lots of free and commercial solutions, some very focused, some feature-rich. An API integrates with your code and gives you access to a range of new features.

Look to Aspose

Aspose are API experts. We create APIs, components and extensions that work independently of Microsoft Automation to extend a platform's native file format manipulation capabilities.

Aspose have developed APIs for .NET, Java, Cloud and Android that lets developers convert, create and manipulate Microsoft Office files – Microsoft Word, Excel, PowerPoint, Visio and Project – and other popular business formats, from PDFs and images to emails. We also have APIs for working with images, barcodes and OCR. The APIs are optimised for stability, speed and ease of use. Our APIs save users weeks, sometimes months, of effort.

Finding the Right Tool

To find the product that's right for you, take a systematic approach:

- List must-have and nice-to-have features.
- Research the market.
- Ask for recommendations.
- Select a few candidates .
- Run trials.
- Evaluate
  - ease of use,
  - support and documentation,
  - performance, and
  - current and future needs.

Aspose.BarCode

A complete toolkit for barcode generation and recognition

- Generate barcodes with customer defined size and color.
- Recognize a large number of barcode types from images.

ASPOSE.BARCODE IS A ROBUST AND RELIABLE BARCODE GENERATION AND RECOGNITION API that allows developers to add barcode generation and recognition functionality to their applications quickly and easily.

Aspose.BarCode supports most established barcode specifications. It can export generated barcodes to multiple image formats, including BMP, GIF, JPED, PNG and TIFF.

Aspose.BarCode gives you full control over every aspect of the barcode image, from background and bar color, through image quality, rotation angle, X-dimension, captions, and resolution.

> Robust and reliable barcode generation and recognition.

Aspose.BarCode can read and recognize most common 1D and 2D barcodes from any image and at any angle. Filters help developers clean up difficult to read images to improve recognition.

Aspose.BarCode offers a large number of symbologies and formatting options.

Common Uses

- Generating and recognizing barcode images.
- Printing barcode labels.
- Enhancing workflow by adding barcode functionality.
- Using recognition functions to drive real-life work processes.

Key Features

- Barcode generation and recognition.
- Comprehensive support for 1D and 2D symbologies.
- Image processing for improved recognition.

Supported File Formats

JPG, TIFF, PNG, BMP, GIF, EMF, WMF, EXIP and ICON.

Format support varies across platforms.

Supported Barcodes

Linear: EAN13, EAN8, UPCA, UPCE, Interleaved2of5, Standard2of5, MSI, Code11, Codabar, EAN14(SCC14), SSCC18, ITF14, Matrix 2 of 5, PZN, Code128, Code39 Extended, Code39 Standard, OPC, Code93 Extended, Code93 Standard, IATA 2 of 5, GS1Code128, ISBN, ISMN, ISSN, ITF6, Pharmacode, DatabarOmniDirectional, VIN, DatabarTruncated, DatabarLimited, DatabarExpanded, PatchCode, Supplement **2D**: PDF417, MacroPDF417, DataMatrix, Aztec, QR, Italian Post 25, Code16K, GS1DataMatrix **Postal**: Postnet, Planet, USPS OneCode, Australia Post, Deutsche Post Identcode, AustralianPosteParcel, Deutsche Post Leticode, RM4SCC, SingaporePost, SwissPostParcel

Platforms

Pricing Info					
	Standard	Enhanced		Standard	Enhanced
Developer Small Business	$599	$1098	Site Small Business	$2995	$5490
Developer OEM	$1797	$3294	Site OEM	$8386	$15372

The pricing info above is for .NET: prices for other platforms may differ. For the latest, contact sales.

Aspose *for* Cloud

The easiest API to Create, Convert & Automate Documents in the cloud.

Convert
Create
Render
Combine
Modify

without installing anything!

Aspose.Words for Cloud

Create and convert docs
Manipulate text
Render documents
Annotate

Aspose.Cells for Cloud

Create spreadsheets
Convert spreadsheets
Manipulate cells and formulas
Render spreadsheets

Aspose.Slides for Cloud

Create presentations
Manage slides
Edit text and images
Read and convert

Aspose.Pdf for Cloud

Create and convert PDFs
Manipulate text, images
Add pages, split, encrypt
Manage stamps

Aspose.Email for Cloud

Create, update, and convert messages
Extract attachments
Use with any language

Aspose.BarCode for Cloud

Generate barcodes
Read barcodes
Set attributes
Multiple image formats

Free Evaluation at www.aspose.com

- +44 141 628 8900
- +1 903 306 1676
  sales@aspose.com
- +61 2 8006 6987

Aspose.Email

Work with emails and calendars without Microsoft Outlook

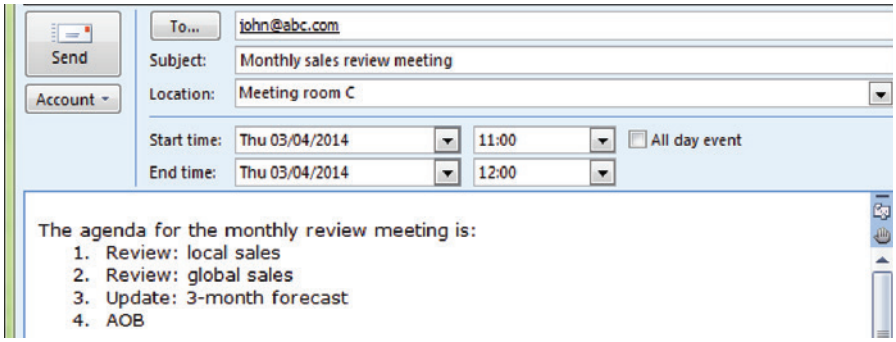
- Complete email processing solution.
- Message file format support.

ASPOSE.EMAIL IS AN EMAIL PROGRAMMING API that allows developers to access and work with PST, EML, MSG and MHT files. It also offers an advanced API for interacting with enterprise mail systems like Exchange and Gmail.

Aspose.Email can work with HTML and plain text emails, attachments and embedded OLE objects. It allows developers to work against SMTP, POP, FTP and Microsoft Exchange servers. It supports mail merge and iCalendar features, customized header and body, searching archives and has many other useful features.

Aspose. Email works with HTML and plain text emails, attachments and embedded OLE objects.

Aspose.Email allows developers to focus on managing email without getting into the core of email and network programming. It gives you the controls you need.



Aspose.Email lets your applications work with emails, attachments, notes and calendars.

Common Uses

- Sending email with HTML formatting and attachments.
- Mail merging and sending mass mail.
- Connecting to POP3 and IMAP mail servers to list and download messages.
- Connecting to Microsoft Exchange Servers to list, download and send messages.
- Create and update tasks using iCalendar.
- Load from and save messages to file or stream (EML, MSG or MHT formats).

Key Features

- A complete email processing solution.
- Support for MSG and PST formats.
- Microsoft Exchange Server support.
- Complete recurrence pattern solution.

Supported File Formats

MSG, MHT, OST, PST, EMLX, TNEF, and EML.

Format support varies across platforms.

Platforms

Pricing Info					
	Standard	Enhanced		Standard	Enhanced
Developer Small Business	$599	$1098	Site Small Business	$2995	$5490
Developer OEM	$1797	$3294	Site OEM	$8386	$15372

The pricing info above is for .NET: prices for other platforms may differ. For the latest, contact sales.

www.aspose.com

EU: +44 141 628 8900

US: +1 903 306 1676
sales@aspose.com

Oceania: +61 2 8006 6987

Aspose.Pdf

Create PDF documents without using Adobe Acrobat

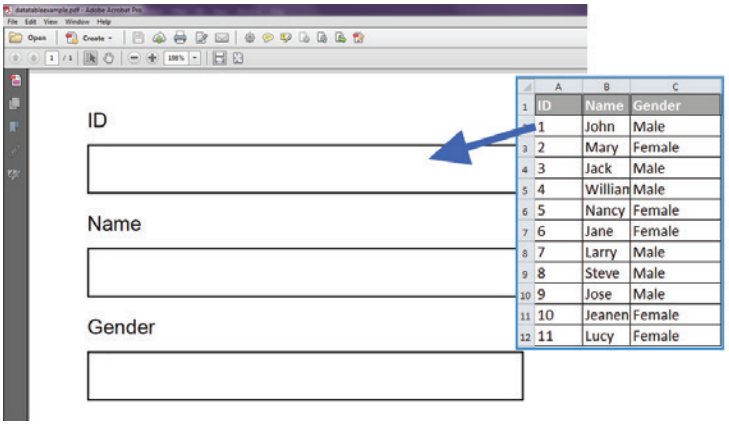
- A complete solution for programming with PDF files.
- Work with PDF forms and form fields.

ASPOSE.PDF IS A PDF DOCUMENT CREATION AND MANIPULATION API that developers use to read, write and manipulate PDF documents without using Adobe Acrobat. Aspose.Pdf is a sophisticated product that integrates with your application to add PDF capabilities.

Aspose.Pdf offers a wealth of features that lets developers compress files, create tables, work with links, add and remove security, handle custom fonts, integrate with external data sources, manage bookmarks, create table of contents, create forms and manage form fields.

> Read, write and manipulate PDF documents independently of Adobe Acrobat.

It helps developers add, work with attachments, annotations and PDF form data, add, replace or remove text and images, split, concatenate, extract or inset pages, and print PDF documents.



Aspose.Pdf can be used to automatically complete PDF forms with external data.

Common Uses

- Creating and editing PDF files.
- Inserting, extracting, appending, concatenating and splitting PDFs.
- Working with text, images, tables, images, headers, and footers.
- Applying security, passwords and signatures.
- Working with forms and form fields.

Key Features

- PDF creation from XML or XSL-FO documents.
- PDF form and field support.
- Advanced security and encryption.
- High-fidelity printing and conversion.
- Supported File Formats
- PDF, PDF/A, PDF/A_1b, PCL, XSL-FO, LaTeX, HTML, XPS, TXT and a range of image formats.

Format support varies across platforms.

Platforms

Pricing Info					
	Standard	Enhanced		Standard	Enhanced
Developer Small Business	$799	$1298	Site Small Business	$3995	$6490
Developer OEM	$2397	$3894	Site OEM	$11186	$18172

The pricing info above is for .NET: prices for other platforms may differ. For the latest, contact sales.

Aspose.Pdf

.Net, Java & Cloud

File Formats

PDF DOC XML XSL-FO XPS HTML BMP JPG PNG ePUB & other image file formats.

Create and Manipulate PDFs

Create new or edit/manipualte existing PDFs.

Form Field Features

Add form fields to your PDFs. Import and export form fields data from select file formats.

Table Features

Add tables to your PDFs with formatting such as table border style, margin and padding info, column width and spanning options, and more.

Get started today at www.aspose.com

Aspose.Note for .NET

Aspose.Note for .NET is an API that lets developers convert Microsoft OneNote pages to a variety of file formats, and extract the text and document information.

Conversion is fast and high-fidelity. The output looks like the OneNote page, no matter how complex the formatting or layout.

Aspose.Note works independently of Office Automation and does not require Microsoft Office or OneNote to be installed.

Modify, convert, render and extract text and images from Microsoft OneNote files without relying on OneNote or other libraries.

Features

File Formats and Conversion		Rendering and Printing	Document Management
Microsoft OneNote 2010, 2010 SP1, 2013	Load, Save	Save as Image (BMP, GIF, JPG, PNG)	• Extract text • Get the number of pages in a document. • Get page information. • Extract images. • Get image information from a document. • Replace text in document.
PDF	Save	Save as PDF	
Images (BMP, GIF, JPG, PNG)	Save		

Aspose.Imaging

Create Images from scratch.

- Load existing images for editing purposes.
- Render to multiple file formats.

ASPOSE.IMAGING IS A CLASS LIBRARY that facilitates the developer to create Image files from scratch or load existing ones for editing purpose. Also, Aspose.Imaging provides the means to save the created or edited Image to a variety of formats. All of the above mentioned can be achieved without the need of an Image Editor. It works independent of other applications and although Aspose.Imaging allows you to save to Adobe PhotoShop® format (PSD), you do not need PhotoShop installed on the machine.

Aspose.Imaging is flexible, stable and powerful. It's many features and image processing routines should meet most imaging requirements. Like all Aspose file format components, Aspose. Imaging introduces support for an advanced set of drawing features along with the core functionality. Developers can draw on Image surface either by manipulating the bitmap information or by using the advanced functionality like Graphics and Paths.

Aspose.Imaging allows creation and manipulation of images.

Create images from scratch. or load existing ones...

Common Uses

- Create images from scratch.
- Load and Edit existing images.
- Export images to a variety of formats.
- Adding watermark to images.
- Export CAD drawings to PDF & raster image formats.
- Crop, resize & RotateFlip images.
- Extract frames from multipage TIFF image.

Key Features

- Create, edit, and save images
- Multiple file formats
- Drawing features
- Export images

Supported File Formats

BMP, JPG, TIFF, GIF, PNG, PSD, DXF, DWG, and PDF.

Platforms

Pricing Info					
	Standard	Enhanced		Standard	Enhanced
Developer Small Business	$399	$898	Site Small Business	$1995	$4490
Developer OEM	$1197	$2694	Site OEM	$5586	$12572

The pricing info above is for .NET.

Aspose.Slides

Work with presentations without using Microsoft PowerPoint

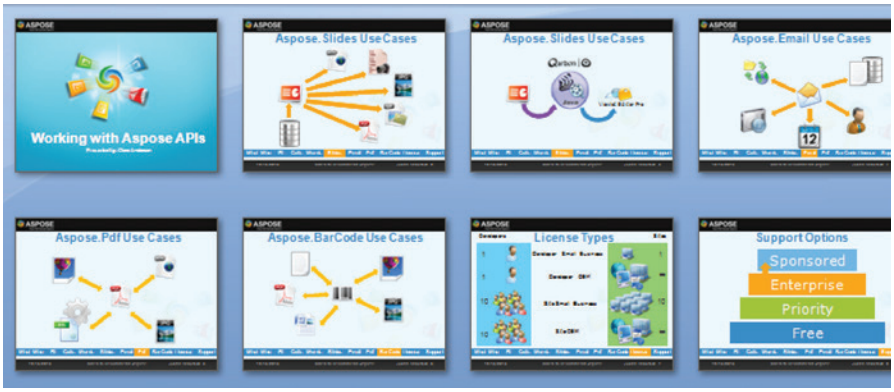
- Complete solution for working with presentation files.
- Export presentations and slides to portable or image formats.

ASPOSE.SLIDES IS A FLEXIBLE PRESENTATION MANAGEMENT API that helps developers read, write and manipulate Microsoft PowerPoint documents. Slides and presentations can be saved to PDF, HTML and image file formats without Microsoft PowerPoint.

Aspose.Slides offers a number of advanced features that make it easy to perform tasks such as rendering slides, exporting presentations, exporting slides to SVG and printing. Developers use Aspose.Slides to build customizable slide decks, add or remove standard graphics and automatically publish presentations to other formats.

> Aspose.Slides gives you the tools you need to work with presentation files.

Aspose.Slides gives developers the tools they need to work with presentation files. It integrates quickly and saves time and money.



Aspose.Slides has advanced features for working with every aspect of a presentation.

Common Uses

- Creating new slides and cloning existing slides from templates.
- Handling text and shape formatting.
- Applying and removing protection.
- Exporting presentations to images and PDF.
- Embedding Excel charts as OLE objects.
- Generate presentations from database.

Key Features

- A complete presentation development solution.
- Control over text, formatting and slide elements.
- OLE integration for embedding external content.
- Wide support for input and output file formats.

Supported File Formats

PPT, HTML, POT, PPS, PPTX, POTX, PPSX, ODP, PresentationML, XPS, PDF and image formats including TIFF and JPG.

Format support varies across platforms.

Platforms

Pricing Info					
	Standard	Enhanced		Standard	Enhanced
Developer Small Business	$799	$1298	Site Small Business	$3995	$6490
Developer OEM	$2397	$3894	Site OEM	$11186	$18172

The pricing info above is for .NET: prices for other platforms may differ. For the latest, contact sales.

Support Services

Get the assistance you need, when you need it, from the people who know our products best.

- Free support for all, even when evaluating
- Get the level of support that suits you and your team

NO ONE KNOWS OUR PRODUCTS AS WELL AS WE DO.
We develop them, support them and use them. Our support is handled through our support forums and is available to all Aspose users.

Support

We are developers ourselves and understand how frustrating it is when a technical issue or a quirk in the software stops you from doing what you need to do. This is why we offer free support. Anyone who uses our product, whether they have bought them or are using an evaluation, deserves our full attention and respect. We have four levels of support that can fit your needs.

> Everyone who uses Aspose products have access to our free support.

Work with the developers that developed and continue to maintain our products.

Support Options

Free

Everyone who uses Aspose products have access to our free support. Our software developers are on stand-by to help you succeed with your project, from the evaluation to roll-out of your solution.

Priority

If you want to know when you'll hear back from us on an issue and know that your issue is prioritized, Priority Support is for you. It provides a more formal support structure and has its own forum that is monitored by our software engineers.

Enterprise

Enterprise customers often have very specific needs. Our Enterprise Support option gives them access to the product development team and influence over the roadmap. Enterprise Support customers have their own, dedicated issue tracking system.

Sponsored

Available to Enterprise customers that would like to request features, this higher prioritized support can ensure your needed features are on our roadmap. A member of our team will produce a feature specification document to capture your requirements and how we intend to fulfill them so the direction development will take is clear up-front.

Pricing Info
To see the Priority and Enterprise support rates, refer to the product price list, or contact our sales team.
Sponsored Support is unique so pricing is specific to each project. Please contact our sales team to discuss.

Aspose has 4 Support Services to best suit your needs

Free Support	Support Forums with no Charge
Priority Support	24 hour response time in the week, issue escalation, dedicated forum
Enterprise Support	Communicate with product managers, influence the roadmap
Sponsored Support	Get the feature you need built now

Technical Support is an issue that Aspose takes very seriously. Software must work quickly and dependably. When problems arise, developers need answers in a hurry. We ensure that our clients receive useful answers and solutions quickly.

Email • Live Chat • Forums

Contact Us

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

ASPOSE
File Format APIs