THE MICROSOFT JOURNAL FOR DEVELOPERS

APRIL 2015 VOL 30 NO 4

msdn magazine

THE MICROSOFT JOURNAL FOR DEVELOPERS

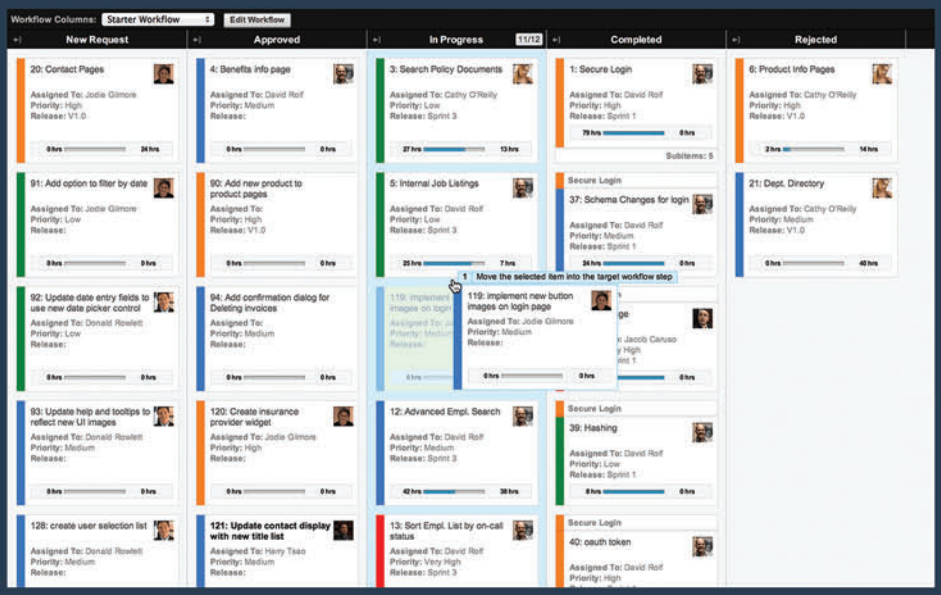
APRIL 2015 VOL 30 NO 4

msdn magazine

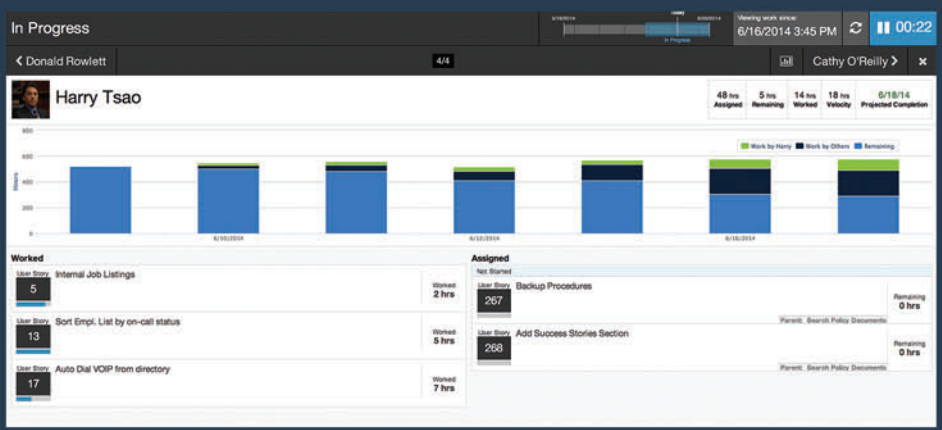
Azure Notification and Event Hubs.....20, 30

COLUMNS

Microsoft



Check out item status and progress at a glance with our Kanban board view



View team members' work capacity and assignments in our Daily Scrum mode

Automated burndown charts, velocity, projected ship dates and more

msdn magazine

APRIL 2015 VOLUME 30 NUMBER 4

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

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

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

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

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

Reaching the Staff

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

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

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

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

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

Microsoft

LEADTOOLS® V19

THE WORLD LEADER IN IMAGING SDKs

Forms Recognition and Processing

Recognize and process structured and unstructured forms including invoices, passports, driver's licenses and checks

- Recognize and extract form fields regardless of image resolution, scale, and other form generation characteristics (OCR, OMR, ICR, 1D & 2D Barcodes)
- Advanced form alignment algorithm compensates for non-linear deformations introduced by different scanners, printers and resolutions
- Automatically detect and correct page orientation and skew angle
- Recognize both vertical and horizontal text from the same document
- Comprehensive confidence reporting for each form field type
- World-class accuracy and speed resulting in significant savings of time and resources

DOWNLOAD OUR 60 DAY EVALUATION
WWW.LEADTOOLS.COM

LEAD TECHNOLOGIES INCORPORATED

SALES@LEADTOOLS.COM
800.637.1840

Editor's Note

MICHAEL DESMOND

Fished Out

I remember one of the first IT articles I ever wrote. I was living in Chicago, writing occasional freelance articles for a weekly computing tabloid while working a temp job for a marketing outfit that shilled cigarettes in bars. Good times. The article was about a floppy disk-borne virus outbreak, which in itself was hardly news. Most malware at the time got onto PCs via the 5.25- or 3.5-inch floppy disks used to move data and install applications. What *was* newsworthy was that the virus showed up on new diskettes sold by a local electronics store.

> But with many searches now conducted over secure HTTPS links, Superfish had a problem. It needed a way into those encrypted conversations.

A quarter-century ago, malware-infected media was a byproduct of regrettable negligence by a manufacturer. Today, manufacturers infect hard drives for fun and profit.

Take the (rather egregious) example of computer maker Lenovo. The company had been outfitting its consumer line of Yoga laptops and convertibles with a pernicious bit of software called Superfish, which Lenovo has described as "visual discovery software," whatever the heck that's supposed to mean. In point of fact, Superfish was a man-in-the-middle (MITM) exploit used to monitor Lenovo users' Web searches and inject its own ads into the results as they were returned back by the remote server. But with many searches now conducted over secure HTTPS links, Superfish had a problem. It needed a way into those encrypted conversations.

Which is when things went *seriously* off the rails at Lenovo. As part of the bundling deal, Lenovo pre-installed into the Windows trusted root store of each of its systems a self-signed, private root certificate from Superfish, essentially imbuing Superfish with all the powers of a certificate authority. The SSL certificates that Superfish presented to intercept traffic were chained to this root certificate, causing the browser to fully trust the certificates Superfish presented.

As Rick Andrews, senior technical director for Trust Services at Symantec, wrote in a blog soon after the Superfish deal blew up: "Pre-installing any root that does not belong to an audited Certificate Authority and marking it as trusted undermines the trust model created and maintained by platform vendors, browser vendors and Certificate Authorities."

It sure does. Worse, the associated private key set up on each computer was encrypted using the same dead-simple password—komodia—which is literally the name of the company that provided the ad injection software for Superfish. I am not making this up. Robert Graham in his Errata Security blog (bit.ly/18mAi00) describes how he was able to quickly extract the Superfish certificate and crack the private key password. Keep in mind, that password is the same for *all* affected Lenovo systems. Any PC with the Superfish certificate installed is vulnerable to having its Web communications intercepted.

Microsoft stepped in quickly, updating its Defender and Security Essentials tools to spot and remove Superfish installations, and Lenovo came to its senses a bit later with a removal tool of its own. None of which resolves the truly galling part of this episode. According to a *Forbes* report (onforb.es/1ErfZeR), Lenovo probably earned between $200,000 and $250,000 in the bundling deal—barely a rounding error on Lenovo's $14.1 billion in revenue in the third quarter. Yet that was enough to motivate a first-line computer manufacturer to completely undermine the root-level security of its paying customers.

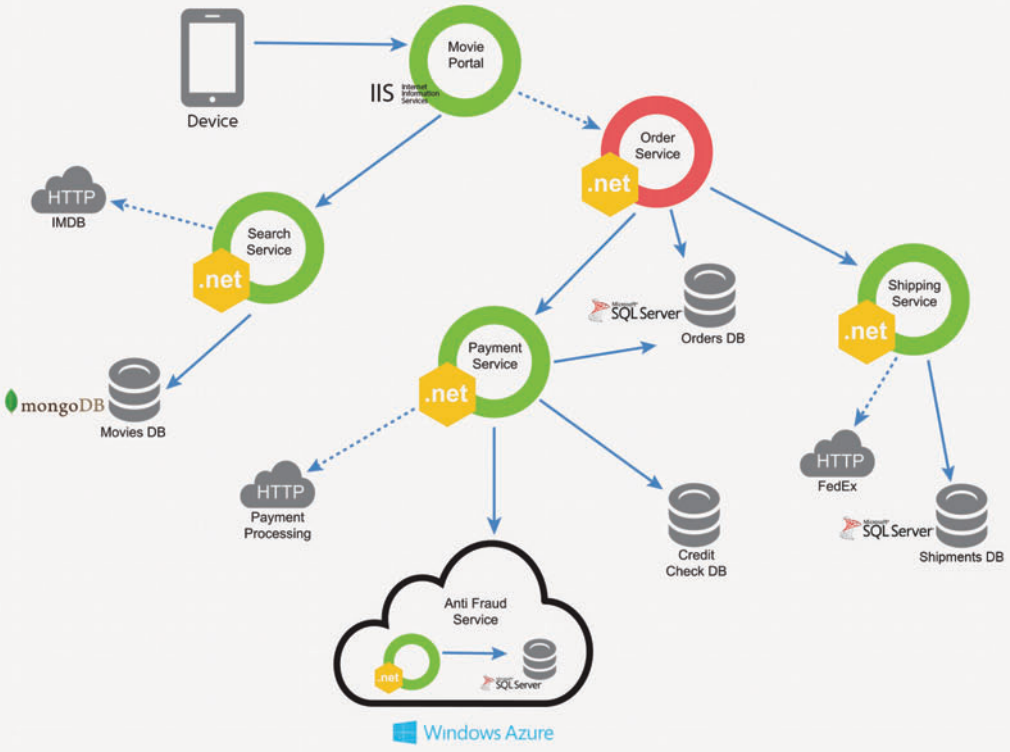
OK, so forget the profit. Maybe manufacturers are infecting hard drives just for the fun of it at this point.

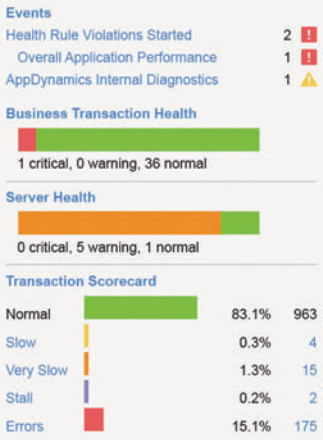
Visit us at msdn.microsoft.com/magazine. Questions, comments or suggestions for *MSDN Magazine*? Send them to the editor: mmeditor@microsoft.com.

APPDYNAMICS

There's nothing about .NET that AppDynamics doesn't see.

AppDynamics gives you the visibility to take command of your .NET application's performance, no matter how complicated or distributed your environment is.





When your business runs on .NET and SQL Server, count on AppDynamics to give you the complete visibility you need to be sure they are delivering the performance and business results you need — no matter how complex, distributed or asynchronous your environment, live 'in production' or during development.

See every line of code. Get a complete view of your environment with deep code diagnostics and auto-discovery. Understand performance trends with dynamic baselining. And drastically reduce time to root cause and remediation.

See why the world's largest .NET deployments rely on the AppDynamics Application Intelligence Platform. Sign up for a FREE trial today at **www.appdynamics.com/Microsoft**.

appdynamics.com

DINO ESPOSITO

Queryable Services

It's getting more common for companies to expose their back-end business services as plain old HTTP endpoints. This type of architecture requires no information about databases and physical data models. Client applications don't even need references to database-specific libraries such as Entity Framework. The physical location of the services is irrelevant and you can keep the back end on-premises or transparently move it to the cloud.

As a solution architect or lead developer, there are two scenarios you should be prepared to face when adopting this strategy. The first is when you have no access whatsoever to the internal workings of the services. In that case, you aren't even in a condition of asking for more or less data to fine-tune the performance of your client application.

> Most consumer API issues concern the quantity and quality of the data returned.

The second scenario is when you're also responsible for maintaining those back-end services and can influence the public API to a certain extent. In this article, I'll focus primarily on the latter scenario. I'll discuss the role of specific technologies to implement queryable services in a flexible way. The technology I'll use is OData services on top of ASP.NET Web API. Nearly everything discussed in this article applies to existing ASP.NET platforms and ASP.NET 5 vNext, as well.

Sealed Back-End Services

Before getting into queryable service design, I'll briefly explore that first scenario in which you have no control over available services. You're given all the details you need to make calls to those services, but have no way to modify the amount and shape of the response.

Such sealed services are sealed for a reason. They're part of the official IT back end of your company. Those services are part of the overall architecture and aren't going to change lightheartedly. As more client applications depend on those services, chances are your company is considering versioning. Generally speaking, though, there has to be a compelling reason before any new release of those services is implemented.

If the sealed API is a problem for the client application you're developing, the only thing you can do is wrap the original services in an additional proxy layer. Then you can use any tricks that serve your purposes, including caching, new data aggregates and inserting additional data. From an architectural standpoint, the resulting set of services then shifts from the infrastructure layer up to the domain services layer. It may even be higher at the application layer (see **Figure 1**).

The Read Side of an API

Modern Web applications are built around an internal API. In some cases, this API becomes public. It's remarkable to consider that ASP.NET 5 vNext pushes an architecture in which ASP.NET MVC and the Razor engine provide the necessary infrastructure for generating HTML views.

ASP.NET Web API represents the ideal infrastructure for handling client requests coming from clients other than browsers and HTML pages. In other words, a new ASP.NET site is ideally devised as a thin layer of HTML around a possibly sealed set of back-end services. The team in charge of the Web application, though, is now also the owner of the back-end API, instead of the consumer. If anyone has an issue with that, you'll hear his complaints or suggestions.

Most consumer API issues concern the quantity and quality of the data returned. The query side of an API is typically the trickiest to create because you never know which way your data is being requested and used in the long run. The command side of an API is usually a lot more stable because it depends on the business domain and services. Domain services do sometimes change, but at least they maintain a different and often slower pace.

Typically, you have a model behind an API. The query side of the API tends to reflect the model whether the API has a REST or an RPC flavor. Ultimately, the sore point of a read API is the format of the data it returns and the data aggregates it supports. This issue has a friendly name—data transfer objects (DTOs).

When you create an API service, you build it from an existing data model and expose it to the outside world through the same native or custom model. For years, software architects devised applications in a bottom-up manner. They always started from the bottom of a typically relational data model. This model traveled all the way up to the presentation layer.

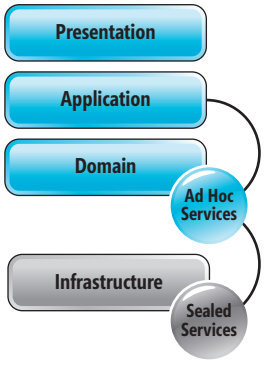


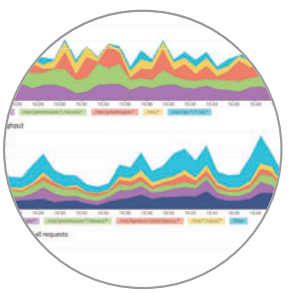
Figure 1 From Sealed Services to More Flexible Application Services

One source of truth

See all your data. Boost performance. Drive accountability for everyone.

Mobile Developers
End-to-end visibility, 24/7 alerting, and crash analysis.

Front-end Developers
Deep insights into your browser-side app's engine.



IT Operations
Faster delivery.
Fewer bottlenecks.
More stability.

App Owners
Track engagement.
Pinpoint issues.
Optimize usability.

Move from finger-pointing blame to data-driven accountability.
Find the truth with a single source of data from multiple views.
newrelic.com/truth

New Relic.

Depending on the needs of the various client applications, some DTO classes were created along the way to ensure that presentation could deal with the right data in the right format. This aspect of software architecture and development is changing today under the effect of the growing role of client applications. While building the command side of a back-end API is still a relatively easy job, devising a single and general enough data model that suits all possible clients is a much harder job. The flexibility of the read API is a winning factor today because you never know which client applications your API will ever face.

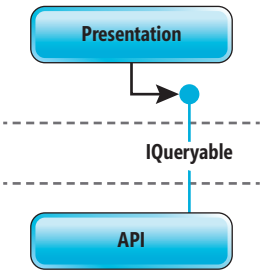


Figure 2 **Queryable Objects Are Queried at the Last Minute**

IQueryable for some IList. Filters you add along the way are simply WHERE clauses added to the actual query being run at some point. As long as you're in the same process space, the amount of data transferred and held in memory is the bare minimum.

How does this affect scalability? The emerging trend for which the vNext platform has been optimized is you keep your Web back end as compact as possible. It will ideally be a single tier. You achieve scalability by replicating the unique tier through a variety of Microsoft Azure Web Roles. Having a single tier for the Web back end lets you use IQueryable everywhere and save yourself a bunch of DTO classes.

Queryable Services

In the latest edition of the book, "Microsoft .NET: Architecting Applications for the Enterprise" (Microsoft Press, 2014), Andrea Saltarello and I formalized a concept we call layered expression trees (LET). The idea behind LET is application layer and domain services exchange IQueryable<T> objects whenever possible. This usually happens whenever the layers live in the same process space and don't require serialization. By exchanging IQueryable<T>, you can defer any required query results from filter composition and data projection to the last minute. You can have these adapted on a per-application basis, instead of some hardcoded form of domain-level API.

The idea of LET goes hand-in-hand with the emerging CQRS pattern. This pushes the separation of the read stack from the command stack. **Figure 2** illustrates the points of having an LET pattern and a bunch of queryable services in your architecture.

The primary benefit of LET is you don't need any DTOs to carry data across layers. You still need to have view model classes at some point, but that's a different story. You have to deal with view model classes as long as you have a UI to fill out with data. View model classes express the desired data layout your users expect. That's the only set of DTO classes you're going to have. Everything else from the level at which you physically query for data and up is served through IQueryable references.

Another benefit of LET and queryable services is the resulting queries are application-level queries. Their logic closely follows domain expert language. This makes it easier to map requirements to code and discuss alleged bugs or misunderstandings with customers. Most of the time, a quick look at the code helps you explain the logic. As an example, here's what an LET query may look like:

```
var model = from i in db.Invoices
                      .ForBusinessUnit(buId)
                      .UnpaidInLast(30.Days)
  orderby i.PaymentDueDate
  select new UnpaidViewModel
    {
      ...
    };
```

From the database context of an Entity Framework root object, you query for all inbound invoices and select those relevant to a given business unit. Among those, you find those still unpaid a number of days past due terms.

The nice thing is IQueryable references are not real data. The query executes against the data source only when you exchange

Implement Queryable Services

In the previous code snippet, I assumed your services are implemented as a layer around some Entity Framework database context. That's just an example, though. You can also fully encapsulate the actual data provider under an ASP.NET Web API façade. That way, you have the benefit of an API that expresses the domain services capabilities and can still reach over HTTP, thus decoupling clients from a specific platform and technology.

Then you can create a Web API class library and host it in some ASP.NET MVC site, Windows service or even some custom host application. In the Web API project, you create controller classes that derive from ApiController and expose methods that return IQueryable<T>. Finally, you decorate each IQueryable method with the EnableQuery attribute. The now obsolete Queryable attribute works, as well. The key factor here is the EnableQuery attribute lets you append OData queries to the requested URL—something like this:

```
[EnableQuery]
public IQueryable<Customer> Get()
{
  return (from c in db.Customers select c);
}
```

> When you create an API service, you build it from an existing data model and expose it to the outside world through the same native or custom model.

This basic piece of code represents the beating heart of your API. It returns no data by itself. It lets clients shape up any return data they want. Look at the code in **Figure 3** and consider it to be code in a client application.

The $select convention in the URL determines the data projection the client receives. You can use the power of OData query syntax to shape up the query. For more on this, see bit.ly/15PVBXv.

For example, one client may only request a small subset of columns. Another client or a different screen in the same client may

Figure 3 **Client Apps Can Shape the Data Returned**

```
var api = "api/customers?$select=LastName";
var request = new HttpRequestMessage()
{
  RequestUri = new Uri(api)),
    Method = HttpMethod.Get,
};
var client = new HttpClient();
var response = client.SendAsync(request).Result;
if (response.IsSuccessStatusCode)
{
  var list = await
    response.Content.ReadAsAsync<IEnumerable<Customer>>();
  // Build view model object here
}
```

query a larger chunk of data. All this can happen without touching the API and creating tons of DTOs along the way. All you need is an OData queryable Web API service and the final view model classes to be consumed. Transferred data is kept to a minimum as only filtered fields are returned.

There are a couple of remarkable aspects to this. First, OData is a verbose protocol and couldn't otherwise be given the role it plays. This means when you apply a $select projection, the JSON payload will still list all fields in the original IQueryable<T> (all fields of the original Customer class in the Get method). However, only the specified fields will hold a value.

Another point to consider is case sensitivity. This affects the $filter query element you use to add a WHERE clause to the query. You might want to call tolower or toupper OData functions (if supported by the OData client library you're using) to normalize comparisons between strings.

Wrapping Up

Quite frankly, I never felt OData was worthy of serious consideration until I found myself—as the owner of a back-end API—in the middle of a storm of requests for different DTOs being returned from the same data model. Every request seemed legitimate and placed for a very noble cause—performance improvement.

At some point, it seemed all those clients wanted to do was "query" the back end in much the same way they would query a regular database table. Then I updated the back-end service to expose OData endpoints, thus giving each client the flexibility to download only the fields in which they were interested.

The type T of each IQueryable method is the key. It may or may not be the same type T you have in the physical model. It can match a plain database table or result from data aggregations done on the server-side and transparent to clients. When you apply OData, though, you let clients query on datasets of a known single entity, T, so why not give it a try? ■

Dino Esposito *is the coauthor of "Microsoft .NET: Architecting Applications for the Enterprise" (Microsoft Press, 2014) and "Programming ASP.NET MVC 5" (Microsoft Press, 2014). A technical evangelist for the Microsoft .NET Framework and Android platforms at JetBrains and a frequent speaker at industry events worldwide, Esposito shares his vision of software at software2cents.wordpress.com and on Twitter at twitter.com/despos.*

Thanks to the following technical expert for reviewing this article: *Jon Arne Saeteras*

Visual C++ 2015 Brings Modern C++ to Legacy Code

Systems programming with Windows relies heavily on opaque handles that represent objects hidden behind C-style APIs. Unless you're programming at a fairly high level, chances are you'll be in the business of managing handles of various kinds. The concept of a handle exists in many libraries and platforms and is certainly not unique to the Windows OS. I first wrote about a smart handle class template back in 2011 (msdn.microsoft.com/ magazine/hh288076) when Visual C++ started introducing some initial C++11 language features. Visual C++ 2010 made it possible to write convenient and semantically correct handle wrappers, but its support for C++11 was minimal and a lot of effort was still required to write such a class correctly. With the introduction of Visual C++ 2015 this year, I thought I'd revisit this topic and share some more ideas about how to use modern C++ to liven up some old C-style libraries.

The best libraries don't allocate any resources and, thus, require minimal wrapping. My favorite example is the Windows slim reader/writer (SRW) lock. Here's all it takes to create and initialize an SRW lock that's ready to use:

```
SRWLOCK lock = {};
```

The SRW lock structure contains just a single void * pointer and there's nothing to clean up! It must be initialized prior to use and the only limitation is that it can't be moved or copied. Obviously, any modernization has more to do with exception safety while the lock is held, rather than resource management. Still, modern C++ can help to ensure these simple requirements are met. First, I can use the ability to initialize non-static data members where they're declared to prepare the SRW lock for use:

```
class Lock
{
  SRWLOCK m_lock = {};
};
```

That takes care of initialization, but the lock can still be copied and moved. For that I need to delete the default copy constructor and copy assignment operator:

```
class Lock
{
  SRWLOCK m_lock = {};

public:

  Lock(Lock const &) = delete;
  Lock & operator=(Lock const &) = delete;
};
```

This prevents both copies and moves. Declaring these in the public part of the class tends to produce better compiler error messages. Of course, now I need to provide a default constructor because one is no longer assumed:

```
class Lock
{
  SRWLOCK m_lock = {};

public:

  Lock() noexcept = default;
  Lock(Lock const &) = delete;
  Lock & operator=(Lock const &) = delete;
};
```

Despite the fact that I haven't written any code, per se—the compiler is generating everything for me—I can now create a lock quite simply:

```
Lock lock;
```

The compiler will forbid any attempts to copy or move the lock:

```
Lock lock2 = lock; // Error: no copy!
Lock lock3 = std::move(lock); // Error: no move!
```

I can then simply add methods for acquiring and releasing the lock in various ways. The SRW lock, as its name suggests, provides both shared reader and exclusive writer locking semantics. **Figure 1** provides a minimal set of methods for simple exclusive locking.

Check out "The Evolution of Synchronization in Windows and C++" (msdn.microsoft.com/ magazine/jj721588) for more information on the magic behind this incredible little locking primitive. All that remains is to provide a bit of exception safety around lock ownership. I certainly wouldn't want to write something like this:

```
lock.Enter();
// Protected code
lock.Exit();
```

Instead, I'd like a lock guard to take care of acquiring and releasing the lock for a given scope:

```
Lock lock;

{
  LockGuard guard(lock);
  // Protected code
}
```

Such a lock guard can simply hold onto a reference to the underlying lock:

```
class LockGuard
{
  Lock & m_lock;
};
```

Like the Lock class itself, it's best that the guard class not allow copies or moves, either:

```
class LockGuard
{
  Lock & m_lock;

public:

  LockGuard(LockGuard const &) = delete;
  LockGuard & operator=(LockGuard const &) = delete;
};
```

All that remains is for a constructor to enter the lock and the destructor to exit the lock. **Figure 2** wraps up that example.

To be fair, the Windows SRW lock is quite a unique little gem and most libraries will require a bit of storage or some kind of resource that must be explicitly managed. I've already shown how best to manage COM interface pointers in "COM Smart Pointers Revisited" (msdn.microsoft.com/magazine/dn904668), so now I'll focus on the more general case of opaque handles. As I wrote previously, a handle class template must provide a way to parameterize not only the type of the handle, but also the way in which the handle is closed, and even what exactly represents an invalid handle. Not all libraries use a null or zero value to represent invalid handles. My original handle class template assumed the caller would provide a handle traits class that gives the necessary semantics and type information. Having written many, many traits classes in the intervening years, I came to realize that the vast majority of these follow a similar pattern. And, as any C++ developer will tell you, patterns are what templates are adept at describing. So, along with a handle class template, I now employ a handle traits class template. The handle traits class template isn't required, but does simplify most definitions. Here's its definition:

```
template <typename T>
struct HandleTraits
{
  using Type = T;

  static Type Invalid() noexcept
  {
    return nullptr;
  }

  // Static void Close(Type value) noexcept;
};
```

Notice what the HandleTraits class template provides and what it specifically does not provide. I've written so many Invalid methods that returned nullptr values that this seemed like an obvious default. On the other hand, each concrete traits class must provide its own Close method for obvious reasons. The comment lives on merely as a pattern to follow. The type alias is likewise optional and is merely a convenience for defining my own traits classes derived from this template. Therefore, I can define a traits class for file handles returned by the Windows CreateFile function, as shown here:

```
struct FileTraits
{
  static HANDLE Invalid() noexcept
  {
    return INVALID_HANDLE_VALUE;
  }

  static void Close(HANDLE value) noexcept
  {
    VERIFY(CloseHandle(value));
  }
};
```

The CreateFile function returns the INVALID_HANDLE_VALUE value if the function fails. Otherwise, the resulting handle must be closed using the CloseHandle function. This is admittedly unusual.

Figure 1 **A Simple and Efficient SRW Lock**

```
class Lock
{
  SRWLOCK m_lock = {};

public:

  Lock() noexcept = default;
  Lock(Lock const &) = delete;
  Lock & operator=(Lock const &) = delete;

  void Enter() noexcept
  {
    AcquireSRWLockExclusive(&m_lock);
  }

  void Exit() noexcept
  {
    ReleaseSRWLockExclusive(&m_lock);
  }
};
```

Figure 2 **A Simple Lock Guard**

```
class LockGuard
{
  Lock & m_lock;

public:

  LockGuard(LockGuard const &) = delete;
  LockGuard & operator=(LockGuard const &) = delete;

  explicit LockGuard(Lock & lock) noexcept :
    m_lock(lock)
  {
    m_lock.Enter();
  }

  ~LockGuard() noexcept
  {
    m_lock.Exit();
  }
};
```

The Windows CreateThreadpoolWork function returns a PTP_WORK handle to represent the work object. This is just an opaque pointer and a nullptr value is naturally returned on failure. As a result, a traits class for work objects can take advantage of the HandleTraits class template, which saves me a bit of typing:

```
struct ThreadPoolWorkTraits : HandleTraits<PTP_WORK>
{
  static void Close(Type value) noexcept
  {
    CloseThreadpoolWork(value);
  }
};
```

So what does the actual Handle class template look like? Well, it can simply rely on the given traits class, infer the type of the handle and call the Close method, as needed. The inference takes the form of a decltype expression to determine the type of the handle:

```
template <typename Traits>
class Handle
{
  using Type = decltype(Traits::Invalid());

  Type m_value;
};
```

This approach keeps the author of the traits class from having to include a type alias or typedef to provide the type explicitly and redundantly. Closing the handle is the first order of business and a safe Close helper method is tucked into the private part of the Handle class template:

```
void Close() noexcept
{
  if (*this) // operator bool
  {
    Traits::Close(m_value);
  }
}
```

This Close method relies on an explicit Boolean operator to determine whether the handle needs to be closed before calling the traits class to actually perform the operation. The public explicit Boolean operator is another improvement over my handle class template from 2011 in that it can simply be implemented as an explicit conversion operator:

```
explicit operator bool() const noexcept
{
  return m_value != Traits::Invalid();
}
```

This solves all kinds of problems and is certainly a lot easier to define than traditional approaches that implement a Boolean-like operator while avoiding the dreaded implicit conversions the

compiler would otherwise permit. Another language improvement, which I've already made use of in this article, is the ability to explicitly delete special members, and I'll do that now for the copy constructor and copy assignment operator:

```
Handle(Handle const &) = delete;
Handle & operator=(Handle const &) = delete;
```

A default constructor can rely on the traits class to initialize the handle in a predictable manner:

```
explicit Handle(Type value = Traits::Invalid()) noexcept :
  m_value(value)
{}
```

And the destructor can simply rely on the Close helper:

```
~Handle() noexcept
{
  Close();
}
```

So, copies aren't permitted, but apart from the SRW lock, I can't think of a handle resource that doesn't permit moving its handle around in memory. The ability to move handles is tremendously convenient. Moving handles involves two individual operations that I might refer to as detach and attach, or perhaps detach and reset. Detach involves releasing the ownership of the handle to the caller:

```
Type Detach() noexcept
{
  Type value = m_value;
  m_value = Traits::Invalid();
  return value;
}
```

The handle value is returned to the caller and the handle object's copy is invalidated to ensure that its destructor doesn't call the Close method provided by the traits class. The complementary attach or reset operation involves closing any existing handle and then assuming ownership of a new handle value:

```
bool Reset(Type value = Traits::Invalid()) noexcept
{
  Close();
  m_value = value;
  return static_cast<bool>(*this);
}
```

The Reset method defaults to the handle's invalid value and becomes a simple way to close a handle prematurely. It also returns the result of the explicit Boolean operator as a convenience. I found myself writing the following pattern quite regularly:

```
work.Reset(CreateThreadpoolWork( ... ));

if (work)
{
  // Work object created successfully
}
```

Here, I'm relying on the explicit Boolean operator to check the validity of the handle after the fact. Being able to condense this into a single expression can be quite convenient:

```
if (work.Reset(CreateThreadpoolWork( ... )))
{
  // Work object created successfully
}
```

Now that I have this handshake in place I can implement the move operations quite simply, beginning with the move constructor:

```
Handle(Handle && other) noexcept :
  m_value(other.Detach())
{}
```

The Detach method is called on the rvalue reference and the newly constructed Handle effectively steals ownership away from the other Handle object. The move assignment operator is only slightly more complicated:

```
Handle & operator=(Handle && other) noexcept
{
  if (this != &other)
  {
    Reset(other.Detach());
  }

  return *this;
}
```

An identity check is first performed to avoid attaching a closed handle. The underlying Reset method doesn't bother to perform this kind of check as that would involve two additional branches for every move assignment. One is prudent. Two is redundant. Of course, move semantics are grand, but swap semantics are even better, especially if you'll be storing handles in standard containers:

```
void Swap(Handle<Traits> & other) noexcept
{
  Type temp = m_value;
  m_value = other.m_value;
  other.m_value = temp;
}
```

Naturally, a non-member and lowercase swap function is required for genericity:

```
template <typename Traits>
void swap(Handle<Traits> & left, Handle<Traits> & right) noexcept
{
  left.Swap(right);
}
```

The final touches to the Handle class template come in the form of a pair of Get and Set methods. Get is obvious:

```
Type Get() const noexcept
{
  return m_value;
}
```

It simply returns the underlying handle value, which may be needed to pass to various library functions. The Set is perhaps less obvious:

```
Type * Set() noexcept
{
  ASSERT(!*this);
  return &m_value;
}
```

This is an indirect set operation. The assertion underscores this fact. I have in the past called this GetAddressOf, but that name disguises or contradicts its true purpose. Such an indirect set operation is needed in cases where the library returns a handle as an out parameter. The WindowsCreateString function is just one example out of many:

```
HSTRING string = nullptr;

HRESULT hr = WindowsCreateString( ... , &string);
```

I could call WindowsCreateString in this way and then attach the resulting handle to a Handle object, or I can simply use the Set method to assume ownership directly:

```
Handle<StringTraits> string;

HRESULT hr = WindowsCreateString( ... , string.Set());
```

That's much more reliable and clearly states the direction in which the data is flowing. The Handle class template also provides the usual comparison operators, but thanks to the language's support for explicit conversion operators, these are no longer necessary for avoiding implicit conversion. They just come in handy, but I'll leave that for you to explore. The Handle class template is just another example from Modern C++ for the Windows Runtime (moderncpp.com). ■

Kenny Kerr *is a computer programmer based in Canada, as well as an author for Pluralsight and a Microsoft MVP. He blogs at kennykerr.ca and you can follow him on Twitter at twitter.com/kennykerr.*

JULIE LERMAN

EF6 Code First Migrations for Multiple Models

Entity Framework 6 introduced support for Code First Migrations to better handle storing data for multiple models in a single database. But the support is very specific and may not be what you imagine. In this article, you'll learn about this feature, what it does and doesn't do, and see how to use it.

Distinct Models, Distinct Tables, Distinct Data: Same Database

EF6 migrations supports migration of multiple models that are completely independent of one another. Both implementations of this feature—using a key to identify a set of migrations or using database schema to group migration histories with the tables for a single model—allow you to store separate, distinct models in the same database.

Not for Sharing Data Across Models or for Multi-Tenant Databases

It's easy to misinterpret the benefits of this feature, so I want to be clear right away on what is *not* supported.

This new multi-model support isn't designed to replicate a single model and across multiple schemas to have a multi-tenant database.

EF6 migrations supports migration of multiple models that are completely independent of one another.

The other pattern many of us hope for when we see this feature is the ability to share a common entity (and its data) across multiple models and map the entity to a single database. That's a very different kind of problem, however, and not one that's easily solved with Entity Framework. I used to try but gave up. I've written previous articles on sharing data across databases in this column (see "A Pattern for Sharing Data Across Domain-Driven Design Bounded Contexts, Part 2" at bit.ly/1817XNT). I also presented a session at TechEd Europe called "Entity Framework Model Partitioning in Domain-Driven Design Bounded Contexts," which was recorded and is available at bit.ly/1AI6xPa.

Pattern One: ContextKey Is the Key

One of the new tools EF6 provides to enable this feature is the ContextKey. This is a new field in the MigrationHistory table of the database that keeps track of every migration. It's partnered with a new property of the same name in the DbMigrationsConfiguration<TContext> class.

By default, the ContextKey will inherit the strongly typed name of the DbMigrationsConfiguration associated with that context. As an example, here's a DbContext class that works with Doctor and Episode types:

```
namespace ModelOne.Context
{
  public class ModelOneContext:DbContext
  {
    public DbSet<Doctor> Doctors { get; set; }
    public DbSet<Episode> Episodes { get; set; }
  }
}
```

As always, the default behavior of the enable-migrations command is to create a DbMigrationsConfiguration class that has the name [YourNamespace].Migrations.Configuration.

When I apply a particular migration (that is, when I call Update-Database in the Visual Studio Package Manager Console), Entity Framework will not only apply the migration, it will also add a new row to the __MigrationHistory table. Here's the SQL for that action:

```
INSERT [dbo].[__MigrationHistory]([MigrationId], [ContextKey], [Model],
  [ProductVersion])
VALUES (N'201501131737236_InitialModelOne',
  N'ModelOne.Context.Migrations.Configuration',
  [hash of the model], N'6.1.2-31219')
```

Notice that the value going into the ContextKey field is ModelOne.Context.Migrations.Configuration, which is the strongly typed name of my DbMigrationsConfiguration<TContext> class.

You can control the ContextKey name by specifying the ContextKey property of the DbMigrationsConfiguration class in the class constructor. I'll rename it to ModelOne:

```
public Configuration()
{
  AutomaticMigrationsEnabled = false;
  ContextKey = "ModelOne";
}
```

Now executing migrations will use ModelOne for the ContextKey field of the Migration table. However, if you've already executed migrations with the default, this will not go well. EF will attempt to reapply all of the migrations, including those that created tables and other database objects, causing the database to throw errors because of the duplicate objects. So my advice is to change that value prior to applying any migrations, otherwise you'll

have to manually update the data in the __MigrationsHistory table.

I've made sure my DbContext type points to a connection string that I've named MultipleModelDb. Rather than rely on the Code First convention to locate a connection string with the same name as the context, I want to have a single connection string I can use for any model that targets this database. I did this by specifying that the context constructor inherit the DbContext overload, which takes a connection string name. Here's the constructor for ModelOneContext:

```
public ModelOneContext()
       : base("MultipleModelDb") {
}
```

Both add-migration and update-database will be able to find the connection string, so I'm assured of migrating the correct database.

Two Contexts, Two ContextKeys

Now that you see how the ContextKey works, let's add in another model with its own ContextKey. I put this model in a separate project. The pattern for doing this when you have multiple models in the same project is a bit different; I'll demonstrate that further along in this article. Here's my new model, ModelTwo:

```
namespace ModelTwo.Context
{
  public class ModelTwoContext:DbContext
  {
    public DbSet<BBCEmployee> BbcEmployees { get; set; }
    public DbSet<HiringHistory> HiringHistories { get; set; }
  }
}
```

ModelTwoContext works with completely different domain classes. Here's its DbConfiguration class, where I specified that the ContextKey be called ModelTwo:

```
internal sealed class Configuration :
DbMigrationsConfiguration<ModelTwoContext>
  {
    public Configuration()
    {
      AutomaticMigrationsEnabled = false;
      ContextKey = "ModelTwo";
    }
```

When I call update-database against the project that contains ModelTwoContext, the new tables are created in the same database and a new row is added to the __MigrationHistory table. This time the ContextKey value is ModelTwo, as you can see in the snippet of SQL that was run by the migration:

```
INSERT [dbo].[__MigrationHistory]([MigrationId], [ContextKey], [Model],
[ProductVersion])
VALUES (N'201501132001186_InitialModelTwo', N'ModelTwo', [hash of the
model], N'6.1.2-31219')
```

As I evolve my domain, my DbContext and my database, EF Migrations will always check back to the relevant set of executed migrations in the __MigrationHistory table using the appropriate ContextKey. That way it will be able to determine what changes to make to the database given the changes I made to the model. This allows EF to correctly manage the migrations for multiple DbContext models that are stored in the database. But remember, it's only able to work because there's no overlap with respect to the database tables to which the two models map.

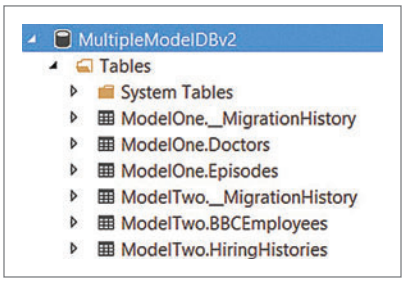


Figure 1 **Migrations and Tables Grouped into Database Schemas**

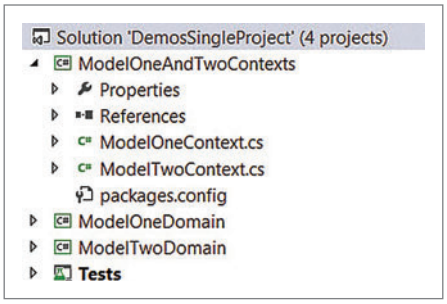


Figure 2 **Placing Multiple DbContext Classes in a Single Project**

Pattern Two: Database Schemas Separate Models and Migrations

The other pattern you can use to allow migrations to work with multiple models in a single database is to separate the migrations and the relevant tables with database schemas. This is as close to simply targeting separate databases as you can get, without some of the resource overhead (such as maintenance and expense) you might incur with multiple databases.

EF6 makes it much easier to define a database schema for a single model by configuring it with a new DbModelBuilder.HasSchema mapping. This will override the default schema name, which, for SQL Server, is dbo.

Remember that even if you don't specify the context key, a default name will be used. So there's no point in removing the context key properties I set to demonstrate how the HasSchema property affects migrations.

I'll set the schema for each of my two context classes in the OnModelCreating method. Here's the relevant code for ModelTwoContext, which I've specified to have a schema named ModelTwo:

```
protected override void OnModelCreating(DbModelBuilder modelBuilder) {
    modelBuilder.HasDefaultSchema("ModelTwo");
}
```

The other context will get the schema name ModelOne.

The result is that all of the database objects to which the ModelTwoContext maps will be in the ModelTwo schema. Additionally, EF will put the __MigrationHistory table for this model in the ModelTwo schema, as well.

To demo this in a clean way, I'm pointing to a different database than I did in the previous examples and applying all of the migrations. Keep in mind that setting the HasDefaultSchema method is a mapping change and requires that you add a new migration to apply that change to the database. **Figure 1** shows the migration and data tables in their separate schemas.

Going forward, whenever you interact with migrations for either context, because they're relegated to their individual schemas, EF won't have a problem maintaining them separately. As a reminder, pay attention to the critical pattern here, which is that there's no overlap with the tables mapped to the two models.

Multiple Models in a Single Project

The two examples you've seen thus far—using the ContextKey or the database schema to separate the model migrations—were set up with each model encapsulated in its own project. This is the way I prefer to architect my solutions. But it's also possible and, in many cases, completely reasonable to have your models in the same

project. Whether you use the ContextKey or the database schema to keep the migrations sorted out, you can achieve this with the addition of only a few extra parameters to the NuGet commands.

For clean separation of these examples, I'll create a new solution with the same classes. I'll keep the domain classes in separate projects, but both models in the same project, as shown in **Figure 2**.

As you know, by default, enable-migrations will create a folder called Migrations for a discovered DbContext in your solution. If you have multiple DbContexts as I do now, enable-migrations will not just randomly select a DbContext for creating migrations; instead, it will return a very helpful message instructing you to use the ContextTypeName parameter to indicate which DbContext to use. The message is so nice that you can just copy and paste from the message to run the necessary commands. Here's the message returned for my project:

```
PM> enable-migrations
More than one context type was found in the assembly 'ModelOne.Context'.
To enable migrations for 'ModelOne.Context.ModelOneContext', use
 Enable-Migrations -ContextTypeName ModelOne.Context.ModelOneContext.
To enable migrations for 'ModelTwo.Context.ModelTwoContext', use
 Enable-Migrations -ContextTypeName ModelTwo.Context.ModelTwoContext.
```

In addition to the –ContextTypeName parameter, I'll add in the MigrationsDirctory parameter to explicitly name the folder to make it easier for me to manage the project assets:

```
Enable-Migrations
-ContextTypeName ModelOne.Context.ModelOneContext
-MigrationsDirectory ModelOneMigrations
```

Figure 3 shows the new folders with their Configuration classes that were created for each migration.

Running Enable-Migrations also adds the code to the DbConfiguration classes, which makes them aware of the directory name. Here's the configuration class for ModelOneContext as an example (the DbConfiguraton file for ModelTwoContext will set its directory name to ModelTwoMigrations, as designated):

```
internal sealed class Configuration : DbMigrationsConfiguration<ModelOne.
Context.ModelOneContext>
  {
    public Configuration()
    {
      AutomaticMigrationsEnabled = false;
      MigrationsDirectory = @"ModelOneMigrations";
    }
```

Because I now have two classes named Configuration, I'll be forced to fully qualify them any time I want to use them. So I'll rename the first to ModelOneDbConfig (as shown in the following code) and the second to ModelTwoDbConfig:

```
internal sealed class ModelOneDbConfig :
DbMigrationsConfiguration<ModelOneContext>
  {
    public ModelOneDbConfig()
      {
        AutomaticMigrationsEnabled = false;
        MigrationsDirectory = @"ModelOneMigrations";
      }
  }
```

You can also specify a ContextKey if you want to override the default, but I'll leave that alone. Remember that I did specify the HasDefaultSchema mapping method in my DbContext classes, so the migration history tables and other database objects will be housed in their own schemas.

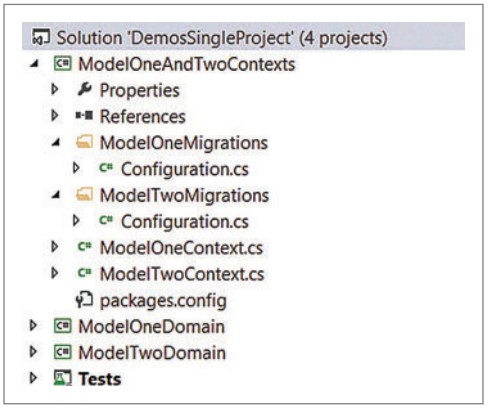


Figure 3 **Result of Specifying the DbContext and Directory Name When Enabling Migrations**

Now it's time to add migrations for both models and apply them to my database. Again, I have to direct EF to which model and migration to use. By pointing to the migration configuration file, EF will know which model to use and in which directory to store the migration files.

Here's my command for adding a migration for ContextOne (remember that I changed the configuration class name so I don't have to use its fully qualified name for ConfigurationTypeName):

```
add-migration Initial
  -ConfigurationTypeName ModelOneDbConfig
```

The resulting migration file gets created in the ModelOneMigrations directory. After I do the same for ModelTwo, I also have a migration file in the ModelTwoMigrations directory.

Now it's time to apply these migrations. I'll need to specify the ConfigurationTypeName again so that EF knows which migration to use. Here's the command for ModelOne:

```
update-database
  -ConfigurationTypeName ModelOneDbConfig
```

I'll run that and then the relevant command for ModelTwo:

```
update-database
  -ConfigurationTypeName ModelTwoDbConfig
```

After running these commands, my database looks just the same as it did in **Figure 1**.

As I modify my models and add and apply migrations, I just need to remember to specify the correct configuration class as a parameter in each of the commands.

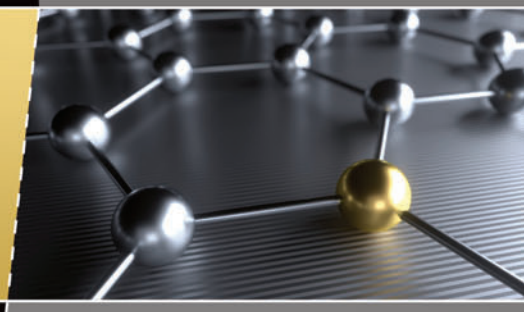
Nice Fit with Domain-Driven Design Modeling

In a recent two-part Data Points column called "A Pattern for Sharing Data Across Domain-Driven Design Bounded Contexts," I wrote about sharing data across domains that are persisted to separate databases. Part One is at bit.ly/1wolxz2 and Part Two is at bit.ly/1817XNT. A number of developers have pointed out that maintaining a separate database on-premises can be a burden and paying for separate databases that are hosted in the cloud can be expensive. The techniques you learned in this article for hosting the tables and data for multiple models in a single database can help you to emulate complete database separation. This new support in EF6 migrations provides a nice solution for those developers. ■

Julie Lerman *is a Microsoft MVP, .NET mentor and consultant who lives in the hills of Vermont. You can find her presenting on data access and other.NET topics at user groups and conferences around the world. She blogs at thedatafarm.com/blog and is the author of "Programming Entity Framework" (2010), as well as a Code First edition (2011) and a DbContext edition (2012), all from O'Reilly Media. Follow her on Twitter at twitter.com/julielerman and see her Pluralsight courses at juliel.me/PS-Videos.*

Thanks to the following Microsoft technical expert for reviewing this article: *Rowan Miller*

Switch to Amyuni PDF



Create & Edit PDFs in .Net - ActiveX - WinRT

- 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 WinRT Component enables publishing C#, C++CX or Javascript apps to Windows Store
- New Postscript/EPS to PDF conversion module

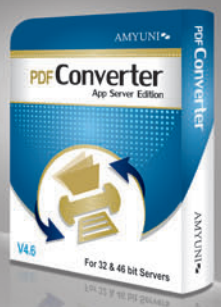
Complete Suite of Accurate PDF Components

- 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

Advanced HTML to PDF & XAML

- Direct conversion of HTML files into PDF and XAML without the use of a web browser or a printer driver
- Easy Integration and deployment within developer's applications
- WebkitPDF is based on the Webkit Open Source library and Amyuni PDF Creator

High Performance PDF Printer For Desktops and Servers



- Our high-performance printer driver optimized for Web, Application and Print Servers. Print to PDF in a fraction of the time needed with other tools. WHQL tested for Windows 32 and 64-bit including Windows Server 2012 R2 and Windows 8.1
- Standard PDF features included with a number of unique features. Interface with any .Net or ActiveX programming language
- Easy licensing and deployment to fit system administrator's requirements

CERTIFIED FOR Windows Server 2008 · Windows 7 · Windows 8 · Windows Server 2012 Certified

AMYUNI

All development tools available at

www.amyuni.com

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

Azure Notification Hubs: Best Practices for Managing Devices

Sara Silva

The mobile applications market is growing faster and faster, and improving the UX of any application is crucial because it increases user loyalty. One of the most important features of modern applications is that they can be kept alive, which means keeping the user informed about the latest events that have occurred in the application, even when it's not being used. This is possible through push notifications.

Each mobile platform has its own push notification service (PNS) responsible for pushing the notifications—short messages—to the device. Windows applications allow apps to receive different push notification types that represent different ways to display the message: toast, tile, raw and badge. For Android applications, on the other hand, only a key/value message is sent to the device and the layout is defined in the application by the class responsible for managing push notifications. In Apple iOS applications, the process is a mix of these approaches.

This article discusses:

- The push notification lifecycle
- Using Notification Hubs
- The two main patterns and two models for managing devices in Notification Hubs
- Using Notification Hubs in Azure Mobile Services

Technologies discussed:

Microsoft Azure, Azure Notificaton Hubs

Code download available at:

github.com/saramgsilva/NotificationHubs

> Azure Notification Hubs is a Microsoft Azure service that provides an easy-to-use infrastructure for sending push notifications from any back end to any mobile platform.

The PNS delivers the notifications, though each application needs a back end or a Web or desktop application to define the message and to connect to the push notification provider to send it.

Azure Notification Hubs is a Microsoft Azure service that provides an easy-to-use infrastructure for sending push notifications from any back end to any mobile platform. Actually, there are two main patterns and two models for managing devices in Notification Hubs. In this article I'll show how each pattern should

Figure 1 **Generic Templates**

```
var toast = new XElement("toast",
  new XElement("visual",
  new XElement("binding",
  new XAttribute("template", "ToastText01"),
  new XElement("text",
  new XAttribute("id", "1"),
  "$(message)"))))).ToString(SaveOptions.DisableFormatting);

var alert = new JObject(
  new JProperty("aps", new JObject(new JProperty("alert", "$(message)"))),
  new JProperty("inAppMessage", notificationText))
  .ToString(Newtonsoft.Json.Formatting.None);

var payload = new JObject(
  new JProperty("data", new JObject(new JProperty("message", "$(message)"))))
  .ToString(Newtonsoft.Json.Formatting.None);
```

be used; discuss the advantages, disadvantages and possible scenarios for each; and describe the different models that can be used. I'll focus on best practices for sending cross-platform and customized push notifications using Notification Hubs, and also show how Notification Hubs is integrated into Azure Mobile Services.

The Push Notification Lifecycle

The push notification lifecycle consists of three main steps:

1. The application makes a request for the push notification service handle, which can be a token, channel URI or a registrationId, depending on the mobile platform.
2. The application sends the PNS handle to the back end to store it.
3. The back end sends a request to the PNS, which then delivers the push notification.

Conceptually this process is very simple, but in practice it's not so easy because the infrastructure required to implement this flow is complex. If an application is provided for different client platforms, it requires an implementation for each one (in reality, it will represent an implementation for each interface provided by each platform). Moreover, the format for each push notification has its own platform specifications and, in the end, these can be hard to maintain. And that's not all: The back end needs to be able to scale some push notification services that don't support broadcasting to multiple devices; to target push notifications to different interest groups; and monitor the push notifications for delivery status. These issues must be handled in the back end and, consequently, this requires a complex infrastructure.

Using Notification Hubs

As I noted, Azure Notification Hubs makes it easy to send mobile push notifications from any back end to any mobile platform, and it supports sending push notifications for different interest groups and providing monitoring, telemetry and scheduling of push notifications. In this way, Notification Hubs is a third-party service that helps send cross-platform and personalized push notifications to applications and implements all the needs of a push notification infrastructure.

To integrate Notification Hubs in an application requires configuration in the Azure Portal; the connection between Notification Hubs and the PNS should be configured in the Configuration separator of the respective Notification Hub. Without these configurations, push notifications can't be sent and an error will occur. (If you're ever in doubt about which push notification services are supported by Notification Hubs, the Configuration separator is the best place to check.) To use Notification Hubs, you'll need to understand tags, templates and the different ways Notification Hubs can be used.

> In general, a tag represents an interest group, which allows you to send push notifications to specific targets.

In general, a tag represents an interest group, which allows you to send push notifications to specific targets. For example, a sports news application could define tags for each sport: cycling, football, tennis and so forth, allowing the user to select what he wants to receive based on his interests. For cases where authentication is required and the focus is the user, the user id can be used as the tag. In practice, a tag is no more than a simple string value (like "cycling," "football" or "tennis"), and it's also useful for localization (for different languages there's a pattern like "en_cycling" or "pt_cycling," which represents tags in English or Portuguese, respectively).

Templates aren't a new concept, but Notification Hubs creates an abstraction that allows you to define either platform-specific templates or generic templates, which means you can specify key/value pairs for each push notification defined when the device is registered. The registration will then provide a native notification (toast, payload or message) that contains expressions (such as $(message)) with a value the back end or application will define when the push notification is sent.

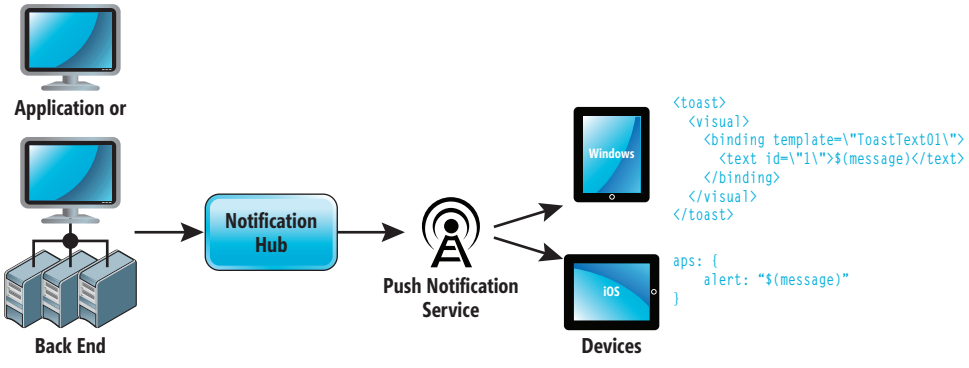


Figure 2 **Sending a Platform-Independent Message**

Figure 1 shows examples of the generic template (for Windows, iOS and Android) using expressions.

The back end or application will fill the value when it sends the key/value (**Figure 2** illustrates the process):

```
{"message", "Message here!"}
```

Registering Devices

In mobile development using Notification Hubs, there are two patterns for managing devices and sending push notifications, which can use two different models. In general, the patterns can be described as follows:

Case 1: Devices connect directly to Notification Hubs

- The client application connects directly to Notification Hubs to register the device.
- The back end or a Web or desktop application connects to Notification Hubs to send the push notification.
- The PNS will deliver the mobile push notification.

Case 2: The back end manages devices in Notification Hubs

- The client application connects to the back end.
- The back end connects with Notification Hubs to register the devices.
- The back end defines the push notification, which will be sent by Notification Hubs.
- The PNS will distribute the mobile push notification.

Both patterns allow you to use either the Registration model or the Installation model; these models describe the way a device sends the required information to the notification hub. The Installation model was introduced recently and is recommended for new applications or even for current applications. This new model doesn't make the Registration model obsolete. Here's a general description of each model:

Registration model: In this model, the application sends a registration request to the Notification Hub, providing the PNS handler and tags, and the hub returns a registration id. For each registration you can choose a native template or a generic template, which will define the message.

In mobile development using Notification Hubs, there are two patterns for managing devices and sending push notifications, which can use two different models.

Installation model: In this model, the application sends an installation request to the Notification Hub, providing all information required for the process: InstallationId (for example, a GUID), tags, PNS handler, templates and secondary templates (for Windows apps).

Although both models are currently available for use, the Installation model was introduced for certain technical reasons, including:

- The installation model is easier to implement and maintain.
- The installation model allows partial updates to modify, add or remove tags, PNS handlers, templates, and so forth, without sending the Installation object; in contrast, the registration model requires the entire registration object.
- The registration model introduces the possibility of duplicate registrations in the back end for the same device.
- The registration model creates complexities in maintaining the list of registrations.
- No matter which pattern you use, the flow will be the same.

The installation model was introduced recently and is recommended for new applications or for adding an event to current applications.

The Installation model can be used in any .NET or Java back end with the Notification Hubs .NET and Java SDKs. For client applications, you can use the REST API with this model, until the new NuGet package that will support the installation model is released. You'll find the current NuGet package with support for the registration model, WindowsAzure.Messaging.Managed, available at bit.ly/1ArliIK.

Now, let's take a look at the two patterns.

Case 1: Devices Connect Directly to Notification Hubs

I'm going to discuss what happens when devices connect directly to Notification Hubs as this is the pattern most frequently used by developers. To describe this, I'll use the registration model.

Registering and unregistering a device in Notification Hubs: The device requests a PNS handle from the PNS, then it connects to a Notification Hub to register, using the PNS handle. The Notification Hub uses this value in the PNS connection.

In practice, for example with Universal Applications, the application starts by requesting the channel from the Windows push notification service (WNS) with the following code:

```
// Get the channel from the application

var pushNotificationChannel =
  await PushNotificationChannelManager.
  CreatePushNotificationChannelForApplicationAsync();
```

For Android, Google Cloud Messaging provides a registrationId, and for iOS, the Apple Push Notification Service provides a token.

A Notification Hub object should be created using the hub name, defined in the Azure Portal, and the connection string (more specifically, the DefaultListenSharedAccessSignature key):

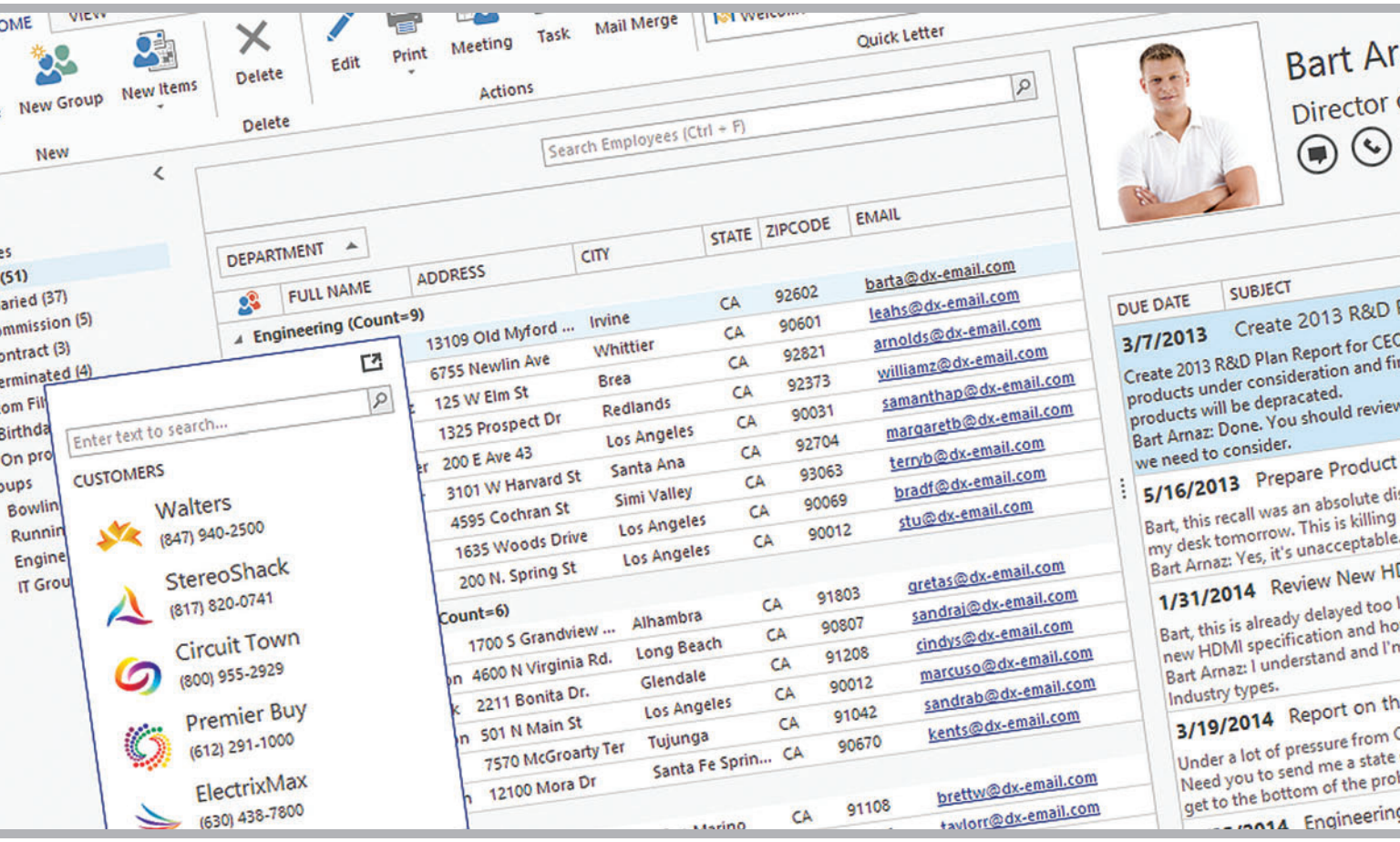
```
// Create the notification hub object
var hub = new NotificationHub(HubName, ConnectionString);
```

Note that the NotificationHub class is provided by the WindowsAzure.Messaging.Managed NuGet package (bit.ly/1ArliIK).

DevExpress®

Your Next Great App Starts Here

With DevExpress tools, you'll create high-performance, high-impact .NET solutions that fully replicate the look, feel and user-experience of Microsoft Office®



Unleash the UI Superhero in You

Download your free 30-day trial today
devexpress.com/try

WIN ASP MVC WPF HTML

Figure 3 **Defining the Native Push Notification**

```
// Define template for Windows
var toast =
  new XElement("toast",
  new XElement("visual",
  new XElement("binding",
  new XAttribute("template", "ToastText01"),
  new XElement("text",
  new XAttribute("id", "1"),
  notificationText)))).ToString(SaveOptions.DisableFormatting);

// Define template for iOS
var alert = new JObject(
  new JProperty("aps", new JObject(new JProperty("alert", notificationText))),
  new JProperty("inAppMessage", notificationText))
  .ToString(Newtonsoft.Json.Formatting.None);

// Define template for Android
var payload = new JObject(
  new JProperty("data", new JObject(new JProperty("message", notificationText)))
  .ToString(Newtonsoft.Json.Formatting.None);
```

Now the application can be registered with the Notification Hub:

```
// Register the device in Notification Hubs
var result =
  await hub.RegisterNativeAsync(pushNotificationChannel.Uri, Tags);
```

When you register a device in Notification Hubs, you don't have to provide a list of tags, but if the application needs to define interest groups, those tags should be stored in the device to use them in a registration update.

Note that the result.RegistrationId received is the id of the registration in Notification Hubs and should not be confused with the registrationId used in Android applications (the id from Google Cloud Messaging).

Sending the push notification: The back end (or an application) connects to the Notification Hub to send the push notification, which can be sent to specific tags (or not) using a specific or generic template.

In this case, it's not important who will connect to the Notification Hub, the back end or a Web or desktop application to send the notification. In practice, a NotificationHubClient object should be created and it will require the hub name, defined in the Azure Portal, and the connection string (more specifically, the DefaultFullSharedAccessSignature key):

```
// Create the Notification Hub client object
var hub = NotificationHubClient.CreateClientFromConnectionString(
  HubName, ConnectionString);
```

Next, the native push notification must be defined, for example, as shown in **Figure 3**.

And then the push notification can be sent:

```
var googleResult =
  await hub.SendGcmNativeNotificationAsync(payload, tags);
var windowsResult =
  await hub.SendWindowsNativeNotificationAsync(toast, tags;
var appleResult =
  await hub.SendAppleNativeNotificationAsync(alert, tags);
```

Whenever a mobile push notification is sent, a list of tags can be provided, but this is optional (it depends on the application requirements).

Distributing the push notification: To finish the process, the PNS delivers the notification to devices. The service will try to push the notification during a limited period of time, which varies for each service.

One of the advantages of this scenario is that it doesn't require a back end, but there's a disadvantage when a user uses more than one device: The tags aren't shared among devices and the user needs to redefine the tags for each one. The mobile application depends on the Notification Hub, and it's needed each time to update the tags in the application, which can be a problem if the user doesn't update to the latest version (a common problem with mobile applications).

Possible scenarios for this case can include:

- The mobile application doesn't require a back end, such as when push notifications are sent by a desktop application with this capacity or by the back office (the admin Web site). For example, consider an application based on an online tech event's news feed that has a settings page where users can subscribe to their interests, such as Developer Events, IT Pro Events and so on. Once the interests are selected, the user will receive push notifications for those interests. Instead of requiring a back end to trigger push notifications, a desktop application or back office can talk directly to the Notification Hub whenever a new event is created to trigger push notifications to all devices that have subscribed to the specific interest.
- The mobile application uses a back end, but the push notifications, for some reason, are not integrated in the services and are sent by a desktop application or by the back office. An example might be a client that has a service that provides information to show in the application, but doesn't allow changes to the service (which can be used by different applications). However, to support push notifications, the client can send notifications using the back office, or even a desktop application.

Case 2: The Back End Manages Devices in Notification Hubs

To describe this pattern I'll use the new approach based on the installation model, introduced recently by the Notification Hubs team.

The application connects to the back end: In this case, the application connects to the back end to create or update the installation object, which will be stored locally.

The device requests the PNS handle from the PNS and gets the last Installation from the Notification Hub, stored on the device. The device then connects to the back end to create or update the installation, from the device, in the Notification Hub. (A new Installation object will be created the first time, but it should be

Figure 4 **Defining a WindowsPushMessage Based on the ToastText01 Template**

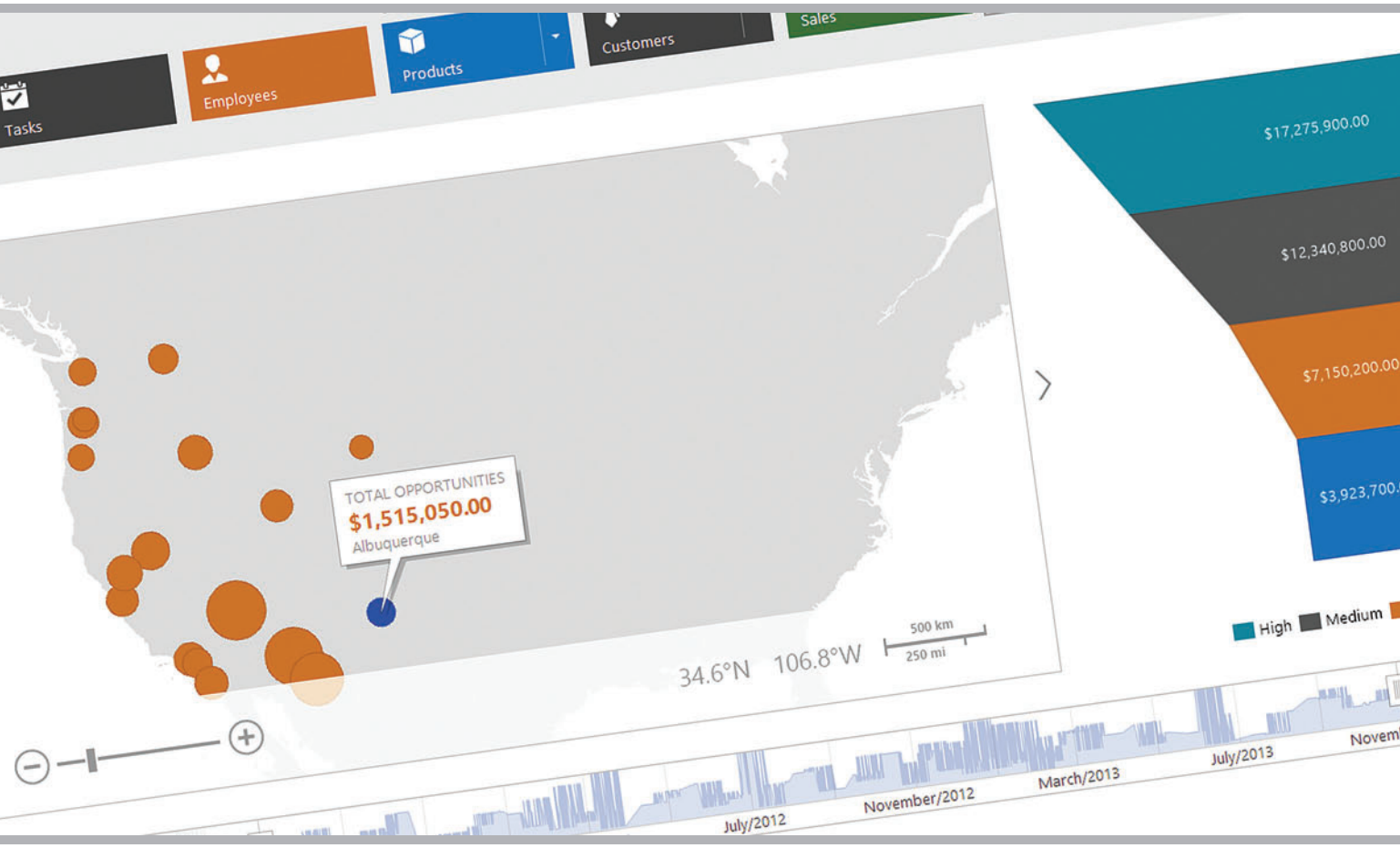
```
public static IPushMessage GetWindowsPushMessageForToastText01(string message)
{
  var payload = new XElement("toast",
    new XElement("visual",
    new XElement("binding",
    new XAttribute("template", "ToastText01"),
    new XElement("text",
    new XAttribute("id", "1"), message))))
    .ToString(SaveOptions.DisableFormatting);

    return new WindowsPushMessage
    {
      XmlPayload = payload
    };
}
```

DevExpress®

Yeah, You Can Touch This

With DevExpress tools, you'll deliver elegant, touch-first apps that are built for today and ready for tomorrow



Unleash the UI Superhero in You

Download your free 30-day trial today
devexpress.com/try

WIN ASP MVC WPF

reused each time the back end connects with the Notification Hub for the same device.)

For example, in Universal Applications, the application will start by requesting the channel URI from the WNS, with the following:

```
// Get the channel from the application
var pushNotificationChannel =
  await PushNotificationChannelManager.
CreatePushNotificationChannelForApplicationAsync();
```

Before making the request to the back end to create or update the installation, the Installation object must be defined:

```
// Retrieve installation from local storage and
// create new one if it does not exist
var installation = SettingsHelper.Installation;
if (installation == null)
{
  installation = new Installation
  {
    InstallationId = Guid.NewGuid().ToString(),
    Platform = NotificationPlatform.Wns,
    PushChannel = pushNotificationChannel.ToString(),
    Tags = new List<string> {"news", "sports"}
  };
}
```

The installation class used in the client application is created from the JSON provided in the documentation.

At this point, you can request the registration with code like this:

```
// Create a client to send the HTTP registration request
var client = new HttpClient();
var request = new HttpRequestMessage(HttpMethod.Post, new Uri(_registerUri))
{
  Content = new StringContent(JsonConvert.SerializeObject(installation),
    Encoding.UTF8, "application/json")
};
var response = await client.SendAsync(request);
```

The back end creates or updates the installation from the device, in the Notification Hub: When the back end receives a request to register the device using the installation model, it connects with the Notification Hub to create or update the installation. For example, in ASP.NET Web API, you can create a NotificationHubController to define the services to manage devices in Notification Hubs. The implementation requires a Notification Hub client object, which can be defined in the constructor, as follows:

```
public class NotificationHubController : ApiController
{
  private readonly NotificationHubClient _notificationHubClient;
  private const string HubName = "<define the notification hub name>";
  private const string ConnectionString = "<define the connection string>"
  public NotificationHubController()
  {
    _notificationHubClient =
      NotificationHubClient.CreateClientFromConnectionString(
      ConnectionString, HubName);
  }
```

Now you can write the method that will create or update the installation, which will receive as input the installation object, like so:

```
[HttpPost]
[Route("api/NH/CreateOrUpdate")]
public async Task CreateOrUpdateAsync(Installation installation)
{
  // Before creating or updating the installation is possible,
  // you must change the tags to have secure tags
  await _notificationHubClient.
    CreateOrUpdateInstallationAsync(installation);
}
```

Note that the tags can be defined and stored in the back end (in both models). The mobile application isn't required to store them or even know about them. For example, consider a bank application that defines tags for each account for a client. When an operation is done for an account, a push notification is sent to the device for that account. In this case, the tags must be secure and only the back end will know them.

In the Installation model, the installation will be stored in the Notification Hub, which means it can be retrieved based on the InstallationId.

The back end defines the push notification, which will be sent by the Notification Hub: The back end is responsible for sending the push notification to the Notification Hub, and you can provide tags and define the template for each platform (when a generic template isn't defined).

The implementation is similar to the scenario in Case 1, where the back end or an application connects to the Notification Hub to send the push notification, so I won't provide any additional code for this case.

Note that the tags can be defined and stored in the back end (in both models).

By the way, in the installation model, it's possible to send a push notification to a specific installation. This is a new feature only for registrations based on the Installation model. Here's the code to do that:

```
_notificationHubClient.SendWindowsNativeNotificationAsync(
  payload, "$InstallationId:{" + installationId + "}");
```

The push notification service will distribute the push notification: To finish the process, the PNS will deliver the push notifications to the devices, during a limited period.

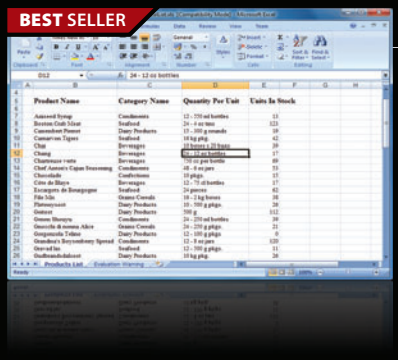
One of the advantages of this case is that tags can be static or dynamic, which means they can change anytime without changing or affecting the mobile applications; tags are secure because each user can only be registered for a tag if he is authenticated; tags can be shared among different devices for the same user; the mobile application is completely independent of Notification Hubs. The disadvantage of this case is the fact the process is more complex than the first case, if the registration model is used.

Possible scenarios for this include:

- A mobile application that connects to a back end and the tags must be secure. A good example for this case is an application related to bank accounts, which supports push notifications to keep users informed about transactions in their accounts.
- A mobile application that connects to a back end and tags must be dynamic. Consider an application that provides information for different music events, using tags to define different events, and for each event users can subscribe to be updated about all related information. The lifecycle for each tag is short and each time a new event is created, a new tag is created for that event. Tags, therefore, are dynamic and can be managed by the back end, keeping the process independent of the mobile application.

In both scenarios, tags should be stored in the back end, but that doesn't mean the mobile client isn't aware of them.

BEST SELLER



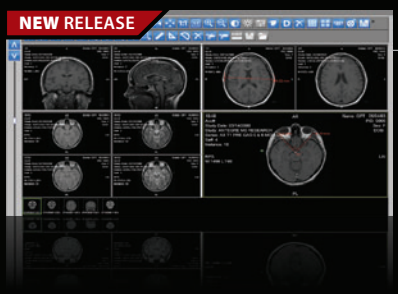
Aspose.Total for .NET | from $2,449.02

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

NEW RELEASE



LEADTOOLS Medical Imaging SDKs V19 | from $4,995.00 SRP

LEAD TECHNOLOGIES

Powerful DICOM, PACS and HL7 functionality.

- Load, save, edit, annotate and display DICOM Data Sets with support for 2014 specifications
- High-level PACS Client and Server components and frameworks
- OEM-ready HTML5 Zero-footprint Viewer and DICOM Storage Server apps with source code
- Medical-specific image processing functions for enhancing 16-bit grayscale images
- Native libraries for .NET, C/C++, HTML5, JavaScript, WinRT, iOS, OS X, Android, Linux & more

NEW RELEASE

GrapeCity ComponentOne Studio 2015 V1 | from $1,315.60

ComponentOne Studio

.NET Tools for Professional Developers.

- Solutions for Data Visualization, Data Management, Reporting and Business Intelligence
- Hundreds of .NET UI controls for all Visual Studio platforms
- Built-in themes and an array of designers for custom styling
- Adaptive, mobile-friendly and touch-enabled controls

BEST SELLER

Help & Manual Professional | from $583.10

ec software

Easily create documentation for Windows, the Web and iPad.

- 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 HTML, WebHelp, CHM, PDF, ePUB, RTF, e-book 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
30 Greyfriars Road
Reading
Berkshire
RG1 1PE
United Kingdom

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

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

GSA Schedule Contract GS-35F-0188R

Updating Registrations

With the registration model, each time you want to update anything related to the registration, such as the tags being used, you must redo the registration; this is required to complete the update. But in the Installation model, you can update just specific details. This means if the application needs to update one tag, the templates or other details, it's possible to do a partial update. Here's the code for updating a tag:

```
// Define the partial update
PartialUpdateOperation replaceTagsOperation =
  new PartialUpdateOperation();
replaceTagsOperation.Operation = UpdateOperationType.Replace;
replaceTagsOperation.Path = "/tags/tennis";
replaceTagsOperation.Value = "cycling";
partialUpdates.Add(replaceTagsOperation);
// Send the partial update
_notificationHubClient.PatchInstallation(
  installationId, partialUpdates);
```

Using Notification Hubs in Azure Mobile Services

Azure Mobile Services allows you to develop an application with a scalable and secure back end (using the Microsoft .NET Framework or Node.js) hosted in Azure. With the focus on mobile applications, Azure Mobile Services provides the main features a mobile application needs, such as CRUD operations, social network authentication, offline support, push notifications and more.

Push notifications are provided in Azure Mobile Services by Notification Hubs, and each time an Azure Mobile Service is created, a Notification Hub is created and associated with the Azure Mobile Service.

Azure Mobile Services allows you to develop an application with a scalable and secure back end (using the Microsoft .NET Framework or Node.js) hosted in Azure.

Mobile applications that use Azure Mobile Services can implement push notifications in a quick and easy way, because the Azure Mobile Services SDK provides APIs for:

- Client applications to register devices using the back end.
- A back end to manage the devices in a Notification Hub; to modify requests from devices, which means the back end can add, modify or delete tags from the request made by mobile applications, or even cancel the registration; to send push notifications (using a specific template and, if tags are required, provide them).

The implementation in the .NET back end will be something similar to:

```
await Services.Push.SendAsync(
  GetWindowsPushMessageForToastText01(
  "My push notification message", tags));
```

The method GetWindowsPushMessageForToastText01 defines the template for the push notification based on a message, and can be implemented as shown in **Figure 4**.

In the client application, for example in Universal Apps, you must request the channel from WNS:

```
// Get the channel
var channel =
  await PushNotificationChannelManager.
  CreatePushNotificationChannelForApplicationAsync();
```

Then the mobile service client should be created, and it will allow interaction with the back end:

```
// Create the MobileServiceClient using the
// AzureEndpoint and admin key
var mobileServiceClient =
  new Microsoft.WindowsAzure.MobileServices.MobileServiceClient(
  AzureEndPoint, AzureMobileServiceApplicationKey);
```

The AzureEndPoint should be something like https://mymobileservice.azure-mobile.net/ and the AzureMobileServiceApplicationKey is a string that represents the application key defined in the Azure Portal.

You can use the method RegisterNativeAsync to register a device, as follows:

```
// Register the device
await MobileServiceClient.Client.GetPush().RegisterNativeAsync(
  channel.Uri, tags);
```

But if you're going to register the device using a generic template, use the RegisterTemplateAsync method.

In Azure Mobile Services (.NET back end), to extend, modify or cancel the registration for Azure Notification Hubs, the INotificationHandler interface should be implemented. By doing so, you can, for example, have secure tags.

Wrapping Up

Notification Hubs provides an abstraction to send push notifications from any back end to any mobile platform, and the new model allows you to define a unique installation that contains the essential information required by Notification Hubs. That installation can be reused, avoiding duplicate registration and complexity, for new applications or even in current applications, the installation model is recommended. There are two main patterns for registering devices to Notifications Hubs, which should be chosen according to the application requirements.

This model lets you send messages focused on specific interest groups using tags, and it's also possible to define generic or specific templates and to scale and monitor the service according to the needs of the application. Finally, easy-to-use SDKs are provided in several languages for both the mobile apps and the back end.

Notification Hubs integrated by default, which is a plus for developers because it enables implementation of the push notification feature in a quick and easy way. ■

Sara Silva *is a mathematics graduate and Microsoft MVP. Nowadays, she works as a mobile developer in Portugal, with a main focus on Windows applications, Xamarin and Microsoft Azure. She blogs at saramgsilva.com and can be followed on Twitter at twitter.com/saramgsilva.*

Thanks to the following Microsoft technical experts for reviewing this article: *Piyush Joshi, Paulo Morgado and Chris Risner*

Work With Relational Data, Not Complex APIs or Services

Whether you are a developer using ADO.NET, JDBC, OData, or MySQL, or a systems integrator working with SQL Server or Biztalk, or even an information worker familiar with ODBC or Excel – our products give you bi-directional access to live data through easy-to-use technologies that you are already familiar with. If you can connect to a database, then you will already know how to connect to Salesforce, SAP, SharePoint, Dynamics CRM, Google Apps, QuickBooks, and much more!

Give RSSBus a try today and see what mean:

visit us online at www.rssbus.com to learn more or download a free trial.

rssbus

INTEGRATION YOUR WAY

Event Hubs for Analytics and Visualization

Bruno Terkaly

There are countless examples of companies that are building solutions based on analytics and visualization. There are many scenarios requiring large-scale data ingestion and analytics. Nobody can deny the huge growth in the social networking space, where tweets from Twitter, posts from Facebook, and Web content from blogs are analyzed in real time and used to provide company's brand awareness. This is the first part of a multi-part series around real-time analytics and visualization, focusing on hyper-scale data flows.

This level of data analysis is big business for companies such as Networked Insights, which helps brands make faster, smarter and more audience-centric decisions. Its marketing solution analyzes and organizes real-time consumer data from the social Web to produce strategic, actionable insights that inform better audience segmentation, content strategy, media investment and brand health. What differentiates its approach is the company's ability to simplify the use of social media data by proactively classifying it across 15,000 consumer-interest dimensions in real time, leveraging computational linguistics, machine learning and other techniques. Some of those consumer interest dimensions include the use of 46 emotional classifiers that go deeper than the roots of positive and negative sentiment and into the nuances of hate, love, desire and fear so marketers understand not only how consumers feel about a product, but what to do next.

This article discusses:

- Data analysis and visualization techniques
- Using the AMQP transmission protocol
- Configuring Azure Event Hubs

Technologies discussed:

Microsoft Azure, Azure Event Hubs, Visual Studio 2013, Ubuntu Linux 14.02

Code download available at:

github.com/brunoterkaly/msdneventhubs

For this level of analysis, the first thing you need is large-scale data ingestion. Using the social media example again, consider that Twitter has hundreds of millions of active users and hundreds of millions of tweets per day. Facebook has an active user base of 890 million. Building the capability to ingest this amount of data in real time is a daunting task.

Once the data is ingested, there's still more work to do. You need to have the data parsed and placed in a permanent data store. In many scenarios, the incoming data flows through a Complex Event Processing (CEP) system, such as Azure Stream Analytics or Storm. That type of system does stream analytics on the incoming data by running standing queries. This might generate alerts or transform the data to a different format. You may then opt to have the data (or part of it) persisted to a permanent store or have it discarded.

That permanent data store is often a JSON-based document database or even a relational database like SQL Server. Prior to being placed into permanent storage, data is often aggregated,

coalesced or tagged with additional attributes. In more sophisticated scenarios, the data might be processed by machine learning algorithms to facilitate making predictions.

Ultimately, the data must be visualized so users can get a deeper understanding of context and meaning. Visualization often takes place on a Web-based dashboard or in a mobile application. This typically introduces the need for a middle tier or Web service, which can expose data from permanent storage and make it available to these dashboards and mobile applications.

This article, and subsequent installments, will take a more straightforward example. Imagine you have thousands of devices in the field measuring rainfall in cities across the United States. The goal is to visualize rainfall data from a mobile device. **Figure 1** demonstrates some of the technologies that might bring this solution to life.

There are several components to this architecture. First is the event producer, which could be the Raspberry Pi devices with attached rainfall sensors. These devices can use a lightweight protocol such as Advanced Message Queuing Protocol (AMQP) to send high volumes of rainfall data into Microsoft Azure Event Hubs. Once Azure has ingested that data, the next tier in the architecture involves reading the events from Azure Event Hubs and persisting the events to a permanent data store, such as SQL Server or DocumentDB (or some other JSON-based store). It might also mean aggregating events or messages. Another layer in the architecture typically involves a Web tier, which makes permanently stored data available to Web-based clients, such as a mobile device. The mobile application or even a Web dashboard then provides the data visualization. This article will focus on the first component.

Event Producer

Because I'm sure most of you typically don't own thousands of Internet of Things (IoT) devices, I've made some simplifications to the example relating to the event producer in **Figure 1**. In this article, I'll simulate the Raspberry Pi devices with Linux-based virtual machines (VMs) running in Azure.

I need to have a C program running on a Linux machine, specifically Ubuntu 14.02. I also need to use AMQP, which was developed in the financial industry to overcome some of the inefficiencies found in HTTP. You can still use HTTP as your transport protocol, but AMQP is much more efficient in terms of latency and throughput. This is a protocol built upon TCP and has excellent performance on Linux.

The whole point in simulating the Raspberry Pi on a Linux VM is you can copy most, if not all, of the code to the Raspberry Pi, which supports many Linux distributions. So instead of reading data from an attached rainfall sensor, the event producer will read from a text file that contains 12 months of rainfall data for a few hundred cities in the United States.

The core technology for high-scale data ingestion in Azure is called Event Hubs. Azure Event Hubs provide hyper-scalable stream ingestion, which lets tens of thousands of Raspberry Pi devices (event producers) send continuous streams of data without interruption. You can scale Azure Event Hubs by defining throughput units (Tus), whereby each throughput unit can handle write operations of 1,000 events per second or 1MB per second (2MB/s for read operations). Azure Event Hubs can handle up to 1 million producers exceeding 1GB/s aggregate throughput.

Sending Events

Sending events to Azure Event Hubs is simple. An entity that sends an event, as you can see in **Figure 1**, is called an event publisher. You can use HTTPS or AMQP for transfer. I'll use Shared Access Signatures (SAS) to provide authentication. An SAS is a time-stamped unique identity providing Send and Receive access to event publishers.

To send the data in C#, create an instance of EventData, which you can send via the Send method. For higher throughput, you

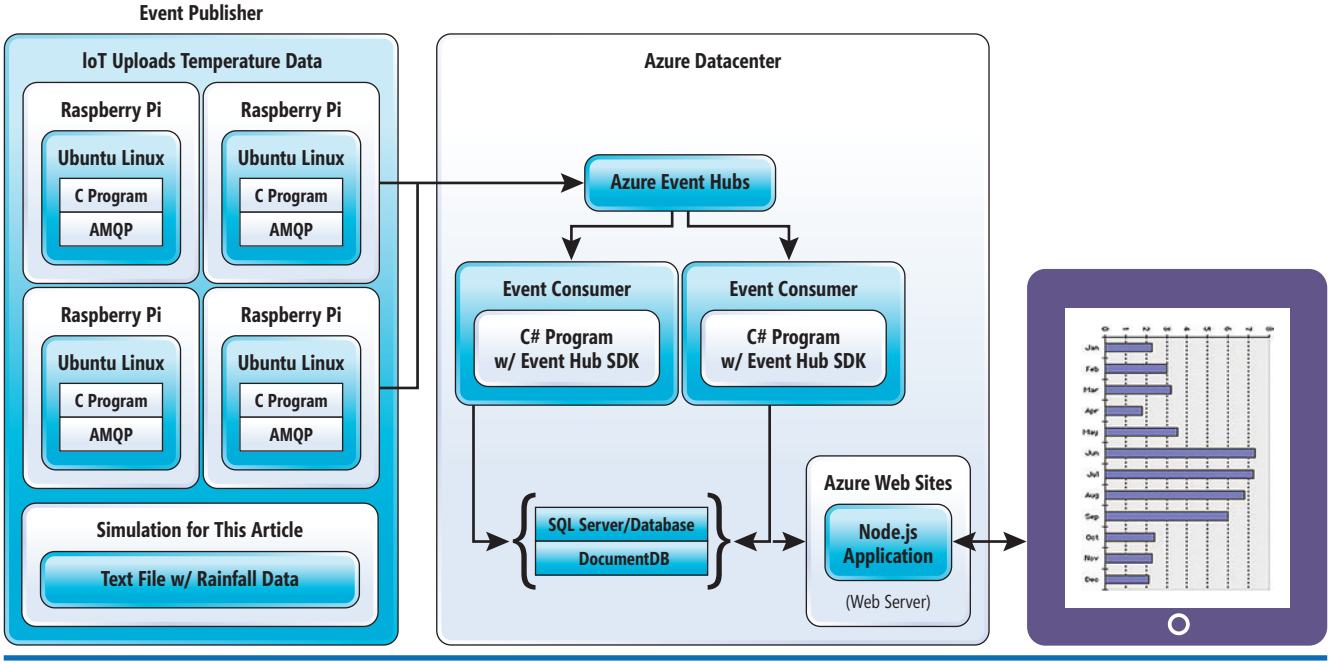


Figure 1 **Overall Architecture of the Rainfall Analysis System**

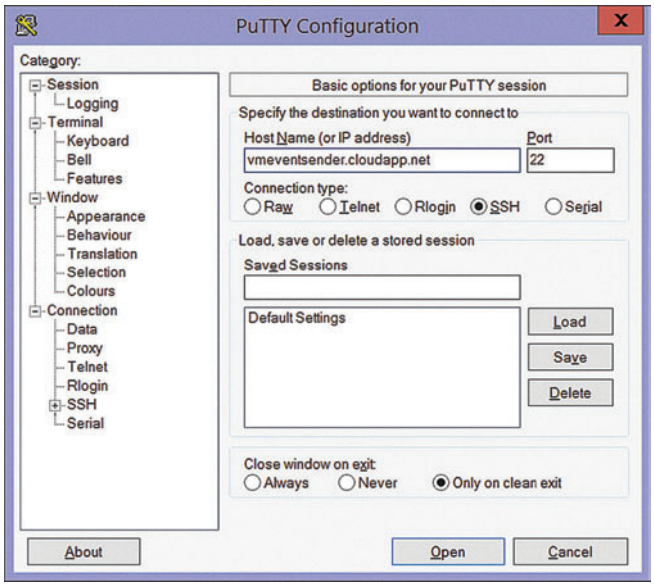


Figure 2 **Use PuTTY to Remote in to the Ubuntu Linux VM**

can use the SendBatch method. In C, AMQP provides a number of methods to provide a similar capability.

Partition Keys

One of the core concepts in Azure Event Hubs is partitions. Partition Keys are values used to map the incoming event data into specific partitions. Partitions are simply an ordered system in which events are held at Event Hubs. As new events are sent, they're added at the end of the partition. You can think of each partition as a separate commit log in the relational database world and work in a similar fashion.

By default, each Azure Event Hub contains eight partitions. You can go beyond 32 partitions, but that requires a few extra steps. You'd only need to do that based on the degree of downstream parallelism required for consuming applications. When publishing events, you can target specific partition keys, but that doesn't scale well and introduces coupling into the architecture.

A better approach is to let the internal hashing function use a round robin event partition mapping capability. If you specify a PartitionKey, the hashing function will assign it to a partition (same PartitionKey always assigns to same partition). The round robin assignment happens only when no partition key is specified.

Get Started

A great place to start is to go through the tutorial at bit.ly/1F2gp9H. You can choose your programming language: C, Java or C#. The code for this article is based on C. Remember, I'm simulating the Raspberry Pi devices by reading from a text file that contains rainfall data. That said, it should be easy to port all this code to a Raspberry Pi device.

Perhaps the best place to start is to provision Azure Event Hubs at the portal. Click on App Services | Service Bus | Event Hub | Quick Create. Part of the provisioning process involves obtaining a shared access signature, which is the security mechanism used by your C program to let it write events into Azure Event Hubs.

Figure 3 **Source Code in C to Send Message to Azure Event Hubs**

```
int main(int argc, char** argv)
{
  printf("Press Ctrl-C to stop the sender process\n");
  FILE * fp;
  char  line[512];
  size_t len = 0;
  size_t read = 0;
  int i = 0;
  int curr_field = 0;
  int trg_col = 0;
  pn_messenger_t *messenger = pn_messenger(NULL);
  pn_messenger_set_outgoing_window(messenger, 1);
  pn_messenger_start(messenger);

  fp = fopen("weatherdata.csv", "r");
  if (fp == NULL)
    exit(EXIT_FAILURE);

  while (fgets(line, 512, fp)!=NULL)
  {
    for (i = 0; line[i] != '\0'; i++)
    {
      if (line[i] == ',')
      {
        fields[curr_field][trg_col] = '\0';
        trg_col = 0;
        curr_field += 1;
      }
      else
      {
        fields[curr_field][trg_col] = line[i];
        trg_col += 1;
      }
    }
    trg_col = 0;
    curr_field = 0;
    for (i = 1; i < 13; i++)
    {
      sendMessage(messenger, i, fields[0], fields[i]);
      printf("%s -> %s\n", fields[0], fields[i]);
    }

    printf("\n");
  }
  fclose(fp);

  // Release messenger resources
  pn_messenger_stop(messenger);
  pn_messenger_free(messenger);

  return 0;
}

int sendMessage(pn_messenger_t * messenger, int month, char *f1, char *f2)
{
  char * address = (char *) "amqps://SendRule:
    [secret key]@temperatureeventhub-ns.servicebus.windows.net/temperatureeventhub";

  int n = sprintf (msgbuffer, "%s,%d,%s", f1, month, f2);
  pn_message_t * message;
  pn_data_t * body;
  message = pn_message();
  pn_message_set_address(message, address);
  pn_message_set_content_type(message, (char*) "application/octect-stream");
  pn_message_set_inferred(message, true);

  body = pn_message_body(message);
  pn_data_put_binary(body, pn_bytes(strlen(msgbuffer), msgbuffer));

  pn_messenger_put(messenger, message);
  check(messenger);
  pn_messenger_send(messenger, 1);
  check(messenger);

  pn_message_free(message);
}
```

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

vslive.com/austin

Austin JUNE 1-4

HYATT REGENCY, AUSTIN, TX

TRACKS INCLUDE:

- Visual Studio / .NET
- JavaScript/HTML5
- ASP.NET
- Mobile Client
- Database and Analytics
- Cloud Computing
- Windows Client

AUSTIN Code Trip — NAVIGATE THE .NET HIGHWAY

DON'T MESS WITH CODE

Register by April 29 and Save $200

USE PROMO CODE VSLAPRTI

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

FLIP OVER FOR DETAILS ON VISUAL STUDIO LIVE! SAN FRANCISCO!

GOLD SPONSOR GitHub

SUPPORTED BY Visual Studio msdn magazine Visual Studio MAGAZINE

PRODUCED BY 1105 MEDIA

```
ABERDEEN- SD,0.48,0.48,1.34,1.83,2.69,3.49,2.92,2.42,1.81,1.63,0.75,0.38
ABILENE- TX,0.97,1.13,1.41,1.67,2.83,3.06,1.69,2.63,2.91,2.9,1.3,1.27
AKRON- OH,2.49,2.28,3.15,3.39,3.96,3.55,4.02,3.65,3.43,2.53,3.04,2.98
ALAMOSA- CO,0.25,0.21,0.46,0.54,0.7,0.59,0.94,1.19,0.89,0.67,0.48,0.33
ALBANY- NY,2.71,2.27,3.17,3.25,3.67,3.74,3.5,3.68,3.31,3.23,3.31,2.76
ALBUQUERQUE- NM,0.49,0.44,0.61,0.5,0.6,0.65,1.27,1.73,1.07,1,0.62,0.49
ALLENTOWN- PA,3.5,2.75,3.56,3.49,4.47,3.99,4.27,4.35,4.37,3.33,3.7,3.39
```

Figure 4 **Partial View of Text File with Rainfall Data**

Once you provision in Azure Event Hubs, you'll also get a URL you'll use to point to your specific instance of Azure Event Hubs inside the Azure datacenter. Because the SAS key contains special characters, you'll need to encode the URL, as directed at bit.ly/1z82c9j.

Provision an Azure VM and Install AMQP

Because this exercise will simulate the IoT scenario by provisioning a Linux-based VM in Azure, it makes sense to do most of your work on a full-featured VM in the cloud. The development and testing environment on a Raspberry Pi is limited. Once you get your code up and running in the VM, you can simply repeat this process and copy the binaries to the Raspberry Pi device. You can find additional guidance on provisioning the Linux-based VM on Azure at bit.ly/1o6mrST. The code base I have used is based on an Ubuntu Linux image.

Now that you've provisioned Azure Event Hubs and a VM hosting Linux, you're ready to install the binaries for the AMQP. As I mentioned earlier, AMQP is a high-performance, lightweight messaging library that supports a wide range of messaging applications such as brokers, client libraries, routers, bridges, proxies and so on. To install AMQP, you'll need to remote into the Ubuntu Linux machine and install the AMQP Messenger Library from bit.ly/1BudbhA.

Because I do most of my work from a Windows machine, I use PuTTY to remote into the Ubuntu image running in Azure. You can download Putty at putty.org. To use PuTTY, you'll need to get your VM URL from the Azure Management Portal. Developers using OS X can simply use SSH to remote in from the Mac terminal.

You might find it easier to install AMQP on CentOS instead of Ubuntu as directed at bit.ly/1F2k47z. Running Linux-based workloads in Azure is a popular tactic. If you're already accustomed to Windows development, you should get familiar with programming in the Linux world. Notice in **Figure 2** that I'm pointing to my Ubuntu VM (vmeventsender.cloudapp.net).

```
Press Ctrl-C to stop the sender process
ABERDEEN- SD -> 0.48
ABERDEEN- SD -> 0.48
ABERDEEN- SD -> 1.34
ABERDEEN- SD -> 1.83
ABERDEEN- SD -> 2.69
ABERDEEN- SD -> 3.49
ABERDEEN- SD -> 2.92
ABERDEEN- SD -> 2.42
ABERDEEN- SD -> 1.81
ABERDEEN- SD -> 1.63
ABERDEEN- SD -> 0.75
ABERDEEN- SD -> 0.38
```

Figure 5 **Output from the Running Code**

Send.c

Once you've installed AMQP on your Ubuntu VM, you can take advantage of the example provided by the AMQP installation process. On my particular deployment, here's where you'll find the send.c file: /home/azureuser/dev/qpid-proton-0.8/examples/messenger/c/send.c. (The code download for this article includes my edited version of send.c.)

You'll essentially replace the default example installed by AMQP with my revised version. You should be able to run it as is, except for the URL pointing to Azure Event Hubs and the encoded shared access signature. The modifications I made to send.c include reading the rainfall data from weatherdata.csv, also included in the code download for this article. The code in **Figure 3** is fairly clear. The main entry method starts by opening the text file, reading one line at a time and parsing the rainfall data into 12 separate pieces—one for each month.

Figure 4 gives you a partial view of the text file with the rainfall data. The C code will connect to the Azure Event Hubs instance you provisioned earlier, using the URL and the encoded shared access signature. Once connected, the rainfall data will load into the message data structure and be sent to Azure Event Hubs using pn_messenger_send. You can get a complete description of these methods at bit.ly/1DzYuud.

There are only two steps remaining at this point. The first is to actually compile the code, which simply involves changing a directory and issuing the make install command. The final step is to actually run the application you just created (see **Figure 5**):

```
// Part 1 - Compiling the code
cd /home/azureuser/dev/qpid-proton-0.8/build/examples/messenger/c
make install
// Part 2 - Running the code
cd /home/azureuser/dev/qpid-proton-0.8/build/examples/messenger/c
./send
```

Wrapping Up

In the next installment, I'll show what it takes to consume the events from Azure Event Hubs and Azure Stream Analytics, and store the data in a SQL database and a JSON-based data store known as Azure DocumentDB. Subsequent articles will delve into exposing this data to mobile applications. In the final article of this series, I'll build a mobile application that provides a visualization of the rainfall data. ■

Bruno Terkaly *is a principal software engineer at Microsoft with the objective of enabling development of industry-leading applications and services across devices. He's responsible for driving the top cloud and mobile opportunities across the United States and beyond from a technology-enablement perspective. He helps partners bring their applications to market by providing architectural guidance and deep technical engagement during the ISV's evaluation, development and deployment. Terkaly also works closely with the cloud and mobile engineering groups, providing feedback and influencing the roadmap.*

Thanks to the following Microsoft technical experts for reviewing this article: *James Birdsall, Pradeep Chellappan, Juan Perez, Dan Rosanova*

Automate Creating, Developing and Deploying Azure Websites

Gyan Jadal

Microsoft Azure is a quick and easy way to create and deploy Web sites or APIs for your cloud consumers. Much has been written on different aspects of Azure Websites, including security, deployment and so on. It's always a challenge to bring this information into a real-life implementation. In this article, I've pulled together these various facets to help you quickly create a production-ready Azure Website. I'll walk you through the end-to-end lifecycle of creating, deploying, configuring and monitoring an Azure Website.

I'll start by creating a Web site with Visual Studio 2013 and deploy it to Azure. Then I'll set up continuous integration from my source code repository to Azure, establishing authentication using the API key or certificates. Finally, I'll show how to create and deploy the site to a production environment using Windows PowerShell.

Create an Azure Web Site

This is the simplest step of the whole process. I'll create an Azure Website and add a controller, skipping the business logic inside it to illustrate the key points. To start, I need the Windows Azure SDK for .NET 2.5, which can be downloaded from bit.ly/1GIhJpu.

This article discusses:

- Creating and configuring an Azure Website
- Setting up authentication
- Setting up automatic deployment for various environments

Technologies discussed:

Microsoft Azure, Visual Studio 2013, Application Insights

Code download available at:

msdn.microsoft.com/magazine/msdnmag0415

Using Visual Studio 2013, I'll create a new ASP.NET Web application by choosing empty template. You can select WebAPI if you want to expose an interface using the same Web site. I select the AppInsights and Hosted in Azure checkboxes. I don't choose any Authentication for now because I'll add it later. I'll also configure Application Insights for deploying to different environments.

When it prompts for Azure Website details, as shown in **Figure 1**, I enter the Site name, select a Region and click OK.

This will set up an empty Web site and publish it to Azure. Now, I'll add some content to the site and use the Open Web Interface for .NET (OWIN) as the middleware. To enable OWIN, I add the following NugGet Packages: Microsoft.Owin.Host.SystemWeb and Microsoft.AspNet.WebApi.Owin.

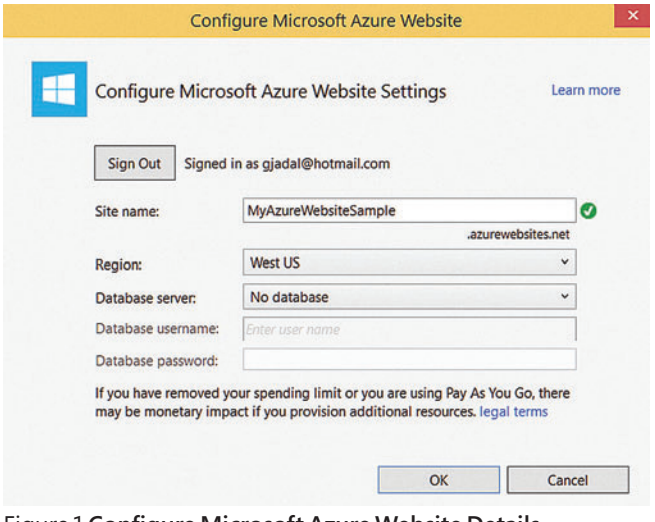


Figure 1 **Configure Microsoft Azure Website Details**

After installing the NuGet packages, I right-click on the Project and select Add OWIN startup class. I'll name the class Startup and add the following code to the Configuration method, which sets up the controller routing:

```
public void Configuration(IAppBuilder app) {
  var config = new HttpConfiguration();
  config.Routes.MapHttpRoute("default", "{controller}/{id}",
    new { id = RouteParameter.Optional});
  app.UseWebApi(config);
}
```

Next, I'll add a controller to provide an API by right-clicking the Project and clicking Add Web API Controller class. I'll name the class MyWebController and leave the generated code alone. Next, I right-click the Project and Publish. In the publish dialog under the Settings tab, I choose the Debug configuration so the site can be debugged remotely. Then I click Next and Publish.

When I navigate to my Azure Website, the controller Get method is invoked by default, which returns a JSON string as ["value1","value2"]. My Web site is up and running on Azure. So far, so good.

Now, I'll add a Client application to invoke the Web site API I just exposed on the cloud. I create a new Console application called MyConsoleClient and add the ASP.NET Web API 2.2 Cient Libraries NuGet Package. This package provides the HttpClient object I'll use to invoke the API. The following code is added to the main method:

```
var httpClient = new HttpClient();
var requestMineType = new MediaTypeWithQualityHeaderValue("application/json");
httpClient.DefaultRequestHeaders.Accept.Add(requestMineType);
var httpResponse =
  httpClient.GetAsync(
  "https://MyAzureWebsiteSample.azurewebsites.net/MyWeb").Result;
Console.WriteLine(httpResponse.Content.ReadAsStringAsync().Result);
Console.ReadLine();
```

Now I'll run the client and see how the controller Get method is invoked and the ["value1","value2"] is printed on the console. I published Debug configuration for my Web site. Although you can debug the site locally, sometimes it's helpful to remotely debug the published Web site while it's still in development. To debug the published site, I go to its Configure tab on the portal and turn on remote debugging, selecting the version of Visual Studio being used (see **Figure 2**).

To debug the Web site remotely and debug my console client, I put a break on the GetAsync line. Next, I open Server Explorer, right-click the Web site and select Attach Debugger, as shown in **Figure 3**.

Now I can step into the code and debug the site remotely from my machine and commit my changes to the source repository.

Set up Continuous Integration

Now I'll work through integrating my site with the Source Repository for Continuous Integration (CI). How cool it would be that, when you check in code to your repository, the Web site project is built and automatically deployed to Azure? The Integrate Source Control feature of Azure Websites gives you an easy way to link your source repository to the Azure Website for continuous integration. On the Azure portal, I navigate to the home tab of the Web site and click on Set up deployment from source control, as shown in **Figure 4**.

Depending on what you're using as your source control provider, you have options to choose from Visual Studio Online, local Git repository, GitHub and so on. As my code is in the Visual Studio

Figure 2 **Enabling Remote Debugging for Microsoft Azure Websites**

Online repository, I choose it on the next screen. Next, I enter my Visual Studio Online account name and authorize my account. I have to provide consent to allow the connection request from Azure to my Visual Studio Online account. Then I choose the Repository name, which is my team project name. After I click Next, the link is established between my Azure Website and Visual Studio Online team project.

In the Builds option in Team Explorer in Visual Studio, I should now see the build definition created for me as MyAzureWebsiteSample_CD so I can right-click and edit the definition. The queue processing is disabled by default on the general tab, so I'll set it to Enabled. In the process parameters under Build/Projects, I browse and select the solution to build.

> Although you can debug the site locally, sometimes it's helpful to remotely debug the published Web site while it's still in development.

At this point, I'll make any other changes to the process parameters in the Process tab. While on the Process tab, take a look at the Deployment parameter. This is where integration is set up for the build definition to the Azure service. If you have multiple projects in the repository, the first Web site as alphabetically ordered will be deployed. To avoid this, ensure you have a separate solution for each Web site for which you have CI enabled. Check in your code or right-click and queue a build. You should see the Web site being built and published to Azure.

You can verify whether continuous integration is working by clicking on the Deployments tab of the Web site, as shown in **Figure 5**. Here, you'll see the last time a successful deployment took place from the CI build.

Configure Authentication

The Azure Website is now ready for development, adding business functionality and reaping the benefits of CI. You'll want to set up authentication for the site so you can authenticate and authorize consumers. There are different scenarios for this setup, details of which you'll find at bit.ly/1CHVni3.

Before I set up my Azure Website for authentication, I'll add Authorize attribute to my controller and publish the site. Running the client to access the site will return the following message:

```
{"Message": "Authorization has been denied for this request."}
```

Here, I'll enable authentication on my Azure Website and update the console client to provide either the API key or certificate credentials for authentication. Authentication and token generation is delegated to Azure Active Directory (AAD). **Figure 6** shows the authentication workflow between the client, AAD and the service.

For AAD Authentication, both the client and Web site should have corresponding application entries registered on AAD. To do this, I navigate to the Active Directory tab on the portal and select the org. directory and click Applications. Next, I select Add an application/Add an application my org. is developing, and enter the name of the Website.

> How cool would it be that, when you check in code to your repository, the Web site project is built and automatically deployed to Azure?

In my case, I enter MyAzureWebsiteSample, then select the Web application or Web API option and click Next. On the next screen, I enter the address to open my site at MyAzureWebsiteSample.azurewebsites.net for the Sign on URL and [mydomain].onmicrosoft.com/MyAzureWebsiteSample as the App ID URI and click OK. This will create an AAD application for the Web site. Next, I add the Client application entry to the AAD, repeating the same steps. For the Sign on URL and App ID URI, I enter http://myconsoleclient.

Because this is a client app, it doesn't have to be a physical URL; a valid URI is sufficient. Now I want to let the client access the MyAzureWebsiteSample application once it has been authenticated. For this, on the Configure tab, in the Permissions to other applications section, I click Add Application, which will open a pop-up window. I select Show Other and search for MyAzureWebsiteSample, then select it and click OK. Back on the Configure tab, in Delegated Permissions is a permission to access MyAzureWebsiteSample (see **Figure 7**). I check it and then click Add application.

Also, I'll note the ClientId for the Client application entry because I'll need it soon. If you want to use API Key for authentication, then a key would be created on this configuration page. Because I'm using certificate authentication, I need to associate my certificate with the Client application entry in AAD. To do this, I'll use the Azure AD Module for Windows PowerShell to automate the process. You can download this at bit.ly/1D4gt8j.

I run the following Windows PowerShell commands to import a .cer file from the local machine and upload the certificate to the client application entry:

```
Connect-msolservice
$cer = New-Object System.Security.Cryptography.X509Certificates.X509Certificate
$cer.Import("$pwd\MyConsoleClientCert.cer")
$binCert = $cer.GetRawCertData()
$credValue = [System.Convert]::ToBase64String($binCert);
New-MsolServicePrincipalCredential -AppPrincipalId "<client id>" -Type
asymmetric -Value $credValue -StartDate $cer.GetEffectiveDateString()
-EndDate $cer.GetExpirationDateString() -Usage verify
```

The clientid in this command is the one I copied in the previous step. To verify the certificate uploaded successfully, I run the following cmdlet providing the clientid:

```
Get-MsolServicePrincipalCredential -AppPrincipalId "<clientid>" -ReturnKeyValues 1
```

This will display all certificates associated with the clientid. In this case, it should display one. I'm done setting up my apps in AAD. The next step is to enable the Web site and client for which I'll use AAD for authentication. First, I'll update MyAzureWebsiteSample project to be set up for AAD. I'll add the following OWIN security NuGet packages related to authentication: Microsoft.Owin.Security and Microsoft.Owin.Security.ActiveDirectory.

In the Startup class of the MyAzureWebsiteSample, I add the following code to the Configuration method:

```
app.UseWindowsAzureActiveDirectoryBearerAuthentication(
  new WindowsAzureActiveDirectoryBearerAuthenticationOptions {
    TokenValidationParameters = new TokenValidationParameters
      {
        ValidAudience = "https://[mydomain].onmicrosoft.com/MyAzureWebsiteSample",
        ValidateAudience = true,
      },
      Tenant = "[mydomain].onmicrosoft.com"
});
```

You'll replace [mydomain] with the name of your AAD directory domain. This will tell the Web site to use AAD authentication for the specified audience URI. Clients accessing the service will need to provide this URI for getting a token from AAD. On the client console, I use the Active Directory Authentication Library, also called ADAL, for authentication logic and add the MyConsoleClient NuGet package to the console application.

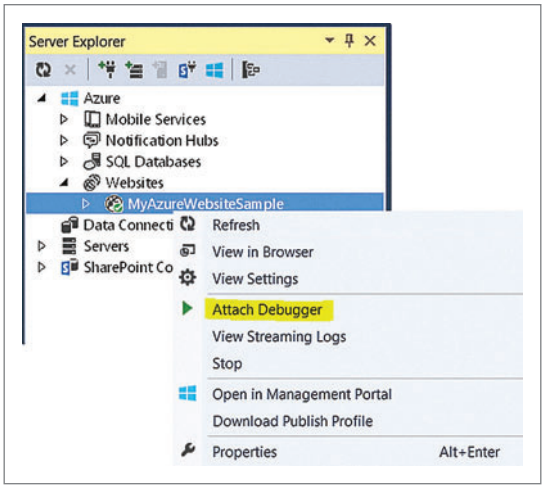


Figure 3 **Attach Debugger to the Published Web site**

ADAL

In the client, I'll use APIs provided by ADAL to acquire a token from AAD by providing the certificate credentials. I replace the previously added code with the code in **Figure 8**, which includes authentication.

You'll replace the clientid and the name of my certificate with your own. I've abstracted the certificate retrieval code into the GetX509CertObject method, which reads the certificate store and creates the X509Certificate object. If you use the APIKey instead of certificate authentication, then

Extreme Performance Linear Scalability

For .NET & Java Apps

(Available on Microsoft Azure & AWS)

Cache data, reduce expensive database trips, and scale your apps to extreme transaction processing (XTP) with NCache.

In-Memory Distributed Cache

- Extremely fast & linearly scalable with 100% uptime
- Mirrored, Replicated, Partitioned, and Client Cache
- Entity Framework & NHibernate Second Level Cache

Open Source

ASP.NET Optimization in Web Farms

- ASP.NET Session State storage
- ASP.NET View State cache
- ASP.NET Output Cache provider

Runtime Data Sharing

- Powerful event notifications for pub/sub data sharing
- Continuous Query events

NCache™

FREE Download

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

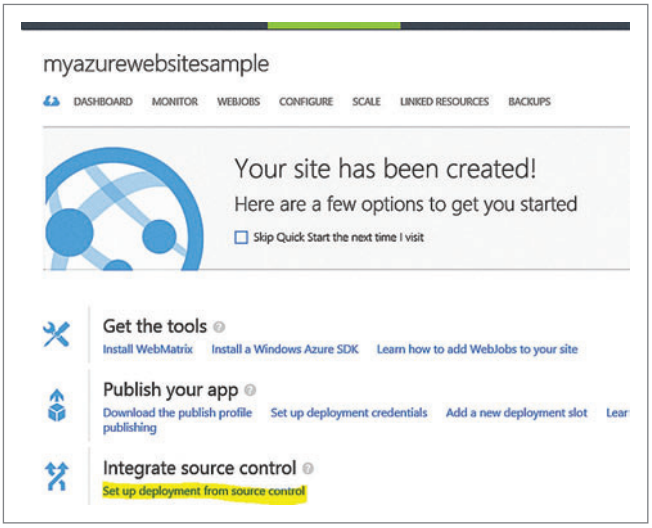


Figure 4 **Set up Deployment from Source Control**

you'll construct a ClientCredential object passing in the clientId and the APIKey noted down from the portal, instead of passing the ClientAssertionCertificate object. If no client credentials are provided to acquire the token, you'll be prompted for AAD tenant user credentials.

I run the code in **Figure 8** to authenticate the client and invoke the MyAzureWebsiteSample controller method and print ["value1", "value2"]. The controller will be presented with the Claims of the client I can use to further authorize the client.

Automate Deployment

So far, I've worked with a development environment to deploy the Azure Website. I just check in the code and the site is deployed, or right-click and it's published—it's that easy.

However, that's not the case for QA or production environments. It may not be feasible to deploy from Visual Studio. Now I'll show you how to automate deployment to any environment using Windows PowerShell. I can perform all these steps from the Portal, as well, but that will involve many manual steps.

Azure PowerShell provides cmdlets to create Azure resources such as Web sites, storage and so on. I'll go over a few of the important aspects of my automation script. The full script is available in the source code download accompanying this article. The resource manager mode in Azure PowerShell lets you logically group multiple resources like Web site, redis cache, SQL Azure and others into a resource group. I use the Switch-AzureMode cmdlet to switch my Azure PowerShell environment between ResourceManager mode and ServiceManagement mode.

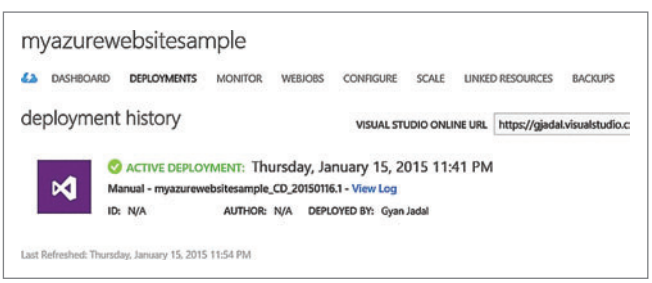


Figure 5 **Deployment History with Continuous Integration**

The following code illustrates a few noteworthy points from the script. First, the Add-AzureAccount displays an interactive window prompting me to sign in to my Azure account. The Select-AzureSubscription sets the current Subscription. Then I switch modes to ResourceManager and create a new ResourceGroup and add my Web site as a new resource:

```
$SubscriptionId = "<subscriptionid>"

Add-AzureAccount
Select-AzureSubscription -SubscriptionId $SubscriptionId -Current
Switch-AzureMode -Name AzureResourceManager

$myResourceGroup = New-AzureResourceGroup -Name "MyResourceGroup"
-Location "West US"

$mywebsite = New-AzureResource -Name "MyAzureWebsiteSample"
  -ResourceType "Microsoft.Web/sites" -ResourceGroupName "MyResourceGroup"
  -Location "West US" -ApiVersion "2014-04-01" -PropertyObject @{}
```

I use the resourcetype as Microsoft.Web/sites. There are other resourcetypes such as Microsoft.Sql/servers/databases you can create with the New-resource cmdlet. In this snippet, I'll create an AzureStorageAccount to associate it to my Web site for storing diagnostics tracing. Here I am not using ResourceManager cmdlets. I created the AzureStorageAccount outside my ResourceGroup. At this time, adding storage accounts to ResourceGroups is not yet supported. To create a table in the storage, I need to obtain the context. I get the context by passing in the storageAccountKey. The storageAccountKey is passed from one cmdlet to another without the user having to know specifics:

```
$myStorage = New-AzureStorageAccount -StorageAccountName "MyStorageAccount"
  -Location "West US"
$storageAccountKey = Get-AzureStorageKey -StorageAccountName "MyStorageAccount"
$context = New-AzureStorageContext -StorageAccountName "MyStorageAccount"
  -StorageAccountKey $storageAccountKey.Primary
$logTable = New-AzureStorageTable -Name $MyStorageTableName -Context $context
```

> Azure PowerShell provides cmdlets to create Azure resources.

The Enable-AzureWebsiteApplicationDiagnostic cmdlet will enable the diagnostics using the storage account I provided:

```
Enable-AzureWebsiteApplicationDiagnostic -Name "MyAzureWebsiteSample"
  -Verbose -LogLevel Information -TableStorage
  -StorageAccountName "MyStorageAccount" -StorageTableName $MyStorageTableName
```

For any Web site, there are some settings in the AppSettings of the web.config that are environment-specific. The following snippet shows how to replace Appsettings for an Azure Web site based on

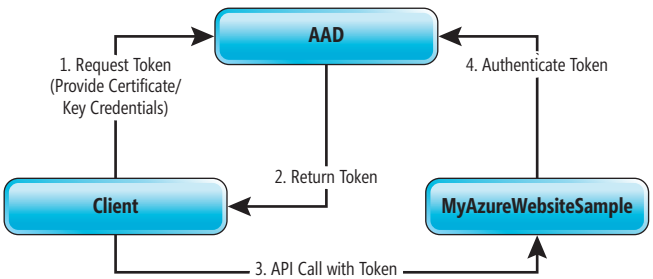
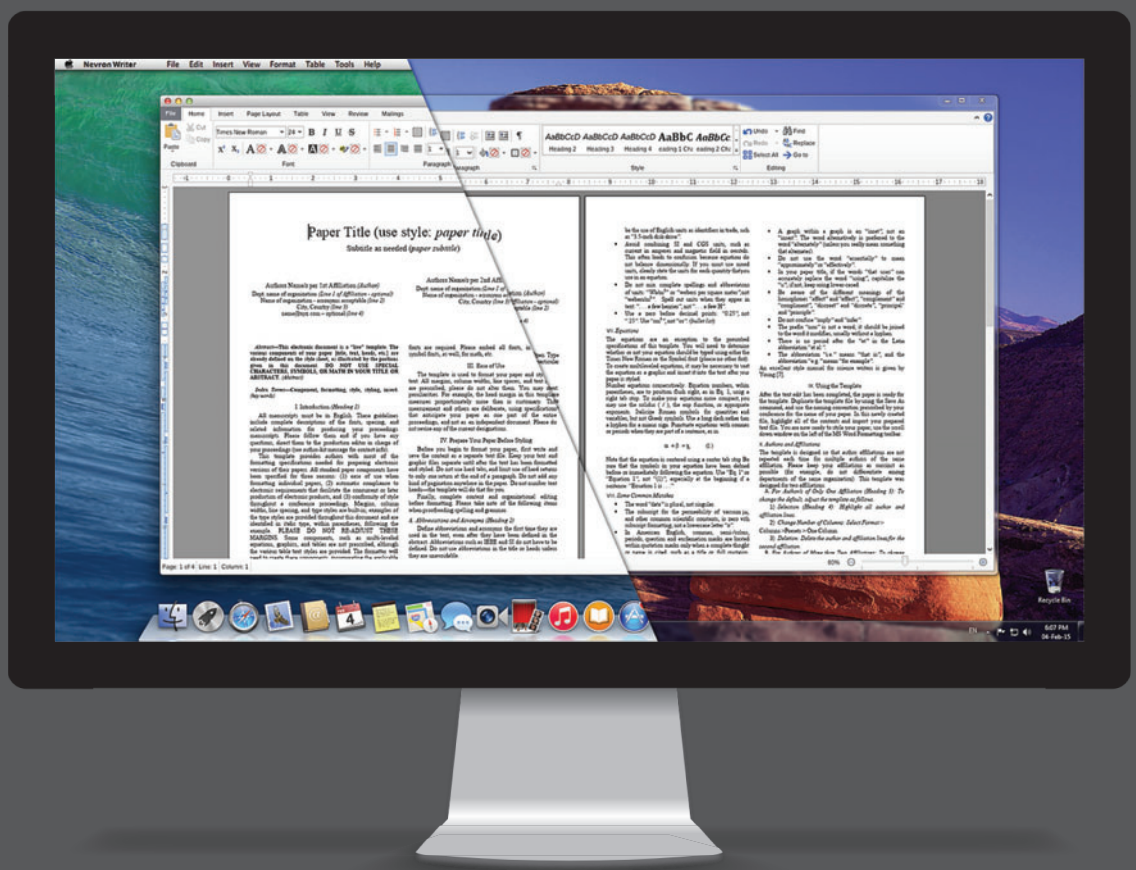


Figure 6 **Azure Active Directory Authentication Workflow**

NOV TEXT EDITOR for .NET

Available for: WinForms | WPF | Silverlight | Xamarin.Mac | MonoMac



MS Word compatible
Open, edit and save DOCX, RTF and HTML files with great accuracy.

Advanced Editing
Edit paragraphs, sections, tables, columns, lists and more.

Embeddable Widgets
Embed 1D and 2D barcodes, gauges, charts etc.

Export to PDF
Save TXT, RTF, DOCX and even HTML files in PDF.

WYSIWYG
100% identical text layout of screen and printed content. Smooth text zooming.

Tables
Supports tables, nested tables and master cell and advanced styling of everything.

Learn more at **www.nevron.com** today

www.nevron.com | email@nevron.com | +1 888-201-6088 (Toll free, USA and Canada)

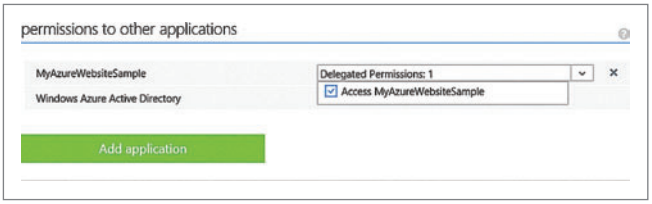


Figure 7 **Setting Permissions on Client Application to Access the Azure Website in Azure Active Directory**

the environment to which it's being deployed. These settings will override the values present in the Web site's web.config:

```
$appSettings = @{
  "CurrentEnv" = "Test";
  "ida:Audience" = "https://[mydomain].onmicrosoft.com/MyAzureWebsiteSample";
  "ida:ClientId" = "8845acba-0820-4ed5-8926-5652s5353s662";
  "ida:Tenant" = "[mydomain].onmicrosoft.com";
  "any:other" = "some other setting";

}

Set-AzureWebsite -Name "MyAzureWebsiteSample" -AppSettings $appSettings
```

A collection of key values are created and set on the Web site by invoking Set-AzureWebsite. This will override the settings in the web.config. You can also change this from the portal later.

So far, I've set up my Web site's infrastructure. Now, I want the application logic in my service to be published. I'll use the Publish-AzureWebsiteProject cmdlet to do this job. I have to provide my sitename and the package produced as part of the build output for my project:

```
Publish-AzureWebsiteProject -Package "<packagepath>" -Name "MyAzureWebsiteSample"
```

I took the package path from my build output:

```
drop\_PublishedWebsites\MyAzureWebsiteSample_Package\MyAzureWebsiteSample.zip
```

The script also uploads the certificate to AAD. You can download the full script from the code download accompanying this article. You can also check bit.ly/1zTBeQo to learn more about the feature-rich experiences the Azure Websites portal provides for managing Web site settings and secrets as you move from development to integration to production deployment environments.

Enable Application Insights

Once your site is in production, you can get the telemetry to understand how the site is behaving in terms of scaling and performance needs. Application Insights provides a telemetry solution for Azure deployed resources. To enable Application Insights, I had to select it as part of the site template. This adds the necessary NuGet packages and an ApplicationInsights.config file (which contains the instrumentation key) to the project. Now when the site is published, the ApplicationInsights resource is created and linked with the key in the config file.

For production environments, create an ApplicationInsights resource on the new portal. At this time, there are no Azure PowerShell cmdlets available to set up the AppInsights on the portal. From the properties, make a note of the Instrumentation key. You need to add this key to the ApplicationInsights.config file before the Web site is published to ensure it's deployed with the site.

However, there's no way to update this file during site publishing. You can get around this issue by not relying on the key from the ApplicationInsights.config file. Instead, add the key to appsettings. This way, you can override it for each environment. How does the site know to use this key? For that, I updated the startup code in my Web site to read the key from the web.config and set it in the App Insights configuration object:

```
Microsoft.ApplicationInsights.Extensibility.
  TelemetryConfiguration.Active.InstrumentationKey = "<read key from appsettings>";
```

I also removed the key from my ApplicationInsights.config file because it's no longer required. Now I pass the key to my automation Azure PowerShell script. This updates the appsettings during site deployment. Once you've set up Application Insights, you can view performance data on the portal or by installing monitoring agents.

Figure 8 **Add Authentication to the API**

```
var httpClient = new HttpClient();
var requestMineType = new MediaTypeWithQualityHeaderValue("application/json");
  httpClient.DefaultRequestHeaders.Accept.Add(requestMineType);

var authContext = new  AuthenticationContext(
  "https://login.windows.net/[mydomain].onmicrosoft.com");

var clientAssertionCertificate = new
ClientAssertionCertificate("<clientId>",
  GetX509CertObject("CN=MyClientConsoleCert"));

var authResult = authContext.AcquireTokenAsync(
  "https://[mydomain].onmicrosoft.com/MyAzureWebsiteSample",
  clientAssertionCertificate).Result;

httpClient.DefaultRequestHeaders.Authorization =
  new AuthenticationHeaderValue("Bearer", authResult.AccessToken);

var httpResponse =
  httpClient.GetAsync(
  "https://MyAzureWebsiteSample.azurewebsites.net/MyWeb").Result;

Console.WriteLine(httpResponse.Content.ReadAsStringAsync().Result);
Console.ReadLine();
```

Azure has come a long way in providing rich and powerful APIs using Azure PowerShell and the new enhanced portal.

Wrapping up

This is a basic step-by-step article for taking an Azure Website from a development environment and setting up authentication and automating deployment in Azure, all the way to a production environment. Azure has come a long way in providing rich and powerful APIs using Azure PowerShell and the new enhanced portal at portal.azure.com.

You can extend this setup to add your own resources such as SQL Azure or additional storage resources. Using the techniques mentioned here, you should be able to quickly spin up a new Azure Website and set up authentication and automation for deployments to any environment. ■

Gyan Jadal *is a software engineer working for Microsoft. He has extensive experience designing and developing solutions for applications based on Azure and enterprise on-premises systems. Reach him at gjadal@microsoft.com.*

Thanks to the following Microsoft technical expert for reviewing this article: *Mani Sengodan*

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

vslive.com

NAVIGATE THE .NET HIGHWAY

Join us on the Ultimate Code Trip in 2015!

Austin

JUNE 1 - 4

HYATT REGENCY

vslive.com/austin

See pages 42–43 for more info

San Francisco

JUNE 15 - 18

THE FAIRMONT

vslive.com/sf

See pages 44-45 for more info

Redmond

AUGUST 10 - 14

MICROSOFT HEADQUARTERS

vslive.com/redmond

See pages 60-61 for more info

New York

SEPTEMBER 28 - OCTOBER 1

NY MARRIOTT AT BROOKLYN BRIDGE

vslive.com/newyork

See pages 68-69 for more info

Orlando

NOVEMBER 16 - 20

LOEWS ROYAL PACIFIC RESORT

CONNECT WITH VISUAL STUDIO LIVE!

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

TURN THE PAGE FOR MORE EVENT DETAILS.

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

Austin AGENDA AT-A-GLANCE

ASP.NET	Cloud Computing	Mobile Client	Database and Analytics	JavaScript / HTML5 Client	Visual Studio / .NET	Windows Client

START TIME	END TIME	Visual Studio Live! Pre-Conference Workshops: Monday, June 1, 2015 (Separate entry fee required)		
9:00 AM	6:00 PM	**M01** Workshop: SQL Server 2014 for Developers - *Leonard Lobel*	**M02** Workshop: Hybrid Mobile Apps with Visual Studio, Cordova, Angular, and Azure - *Brian Noyes*	**M03** Workshop: ALM and DevOps with the Microsoft Stack - *Brian Randell*
6:45 PM	9:00 PM	Dine-A-Round with Speakers		

START TIME	END TIME	Visual Studio Live! Day 1: Tuesday, June 2, 2015			
8:00 AM	9:00 AM	**KEYNOTE:** The Future of Application Development - Visual Studio 2015 and .NET 2015 - *Jay Schmelzer, Director of Program Management, Visual Studio Team, Microsoft*			
9:15 AM	10:30 AM	**T01** Building Mobile Cross-Platform Apps with C# and Xamarin - *Nick Landry*	**T02** UX Design Principle Fundamentals for Non-Designers - *Billy Hollis*	**T03** Azure 10-10: Top 10 Azure Announcements in "T-10" Months - *Vishwas Lele*	**T04** A Lap Around Visual Studio 2015 - *Robert Green*
10:45 AM	12:00 PM	**T05** Building Mobile Cross-Platform Apps in C# with Azure Mobile Services - *Nick Landry*	**T06** Designing and Building UX for Finding and Visualizing Data in XAML Applications - *Billy Hollis*	**T07** Cloud or Not, 10 Reasons Why You Must Know "Websites" - *Vishwas Lele*	**T08** Putting CodedUI Tests on Steroids - *Donovan Brown*
12:00 PM	1:30 PM	Lunch — Visit Exhibitors			
1:30 PM	2:45 PM	**T09** ASP.NET 5 in All Its Glory - *Adam Tuliper*	**T10** Getting Started with Universal Apps for Windows and Windows Phone - *Philip Japikse*	**T11** Microsoft Azure SQL Database: SQL Server in the Cloud - *Leonard Lobel*	**T12** Moving DevOps to the Cloud Using Visual Studio Online, Release Management Online, and Azure - *Donovan Brown*
3:00 PM	4:15 PM	**T13** Hack Proofing Your Web Applications - *Adam Tuliper*	**T14** Building Windows 10 LOB Apps - *Robert Green*	**T15** Database Development with SQL Server Data Tools - *Leonard Lobel*	**T16** What's New in ALM - *Brian Randell*
4:15 PM	5:30 PM	Welcome Reception			

START TIME	END TIME	Visual Studio Live! Day 2: Wednesday, June 3, 2015			
8:00 AM	9:00 AM	**KEYNOTE:** User Interaction Design in a NUI World - *Tim Huckaby, Founder/Chairman - InterKnowlogy & Actus Interactive Software*			
9:15 AM	10:30 AM	**W01** Implementing M-V-VM (Model-View-View Model) for WPF - *Philip Japikse*	**W02** AngularJS 101 - *Deborah Kurata*	**W03** Moving Web Apps to the Cloud - *Eric D. Boyd*	**W04** To Git or Not to Git for Enterprise Development - *Benjamin Day*
10:45 AM	12:00 PM	**W05** Everything You Always Wanted To Know About REST (But Were Afraid To Ask) - *Jon Flanders*	**W06** AngularJS Forms and Validation - *Deborah Kurata*	**W07** Solving Security and Compliance Challenges with Hybrid Clouds - *Eric D. Boyd*	**W08** Load Testing ASP.NET & WebAPI with Visual Studio - *Benjamin Day*
12:00 PM	1:30 PM	Birds-of-a-Feather Lunch — Visit Exhibitors			
1:30 PM	2:45 PM	**W09** Building Cross Platform UI with Xamarin.Forms - *Walt Ritscher*	**W10** Securing Angular Apps - *Brian Noyes*	**W11** To Be Announced	**W12** Automated Build, Test & Deploy with TFS, ASP.NET, and SQL Server - *Benjamin Day*
3:00 PM	4:15 PM	**W13** iBeacons and Contextual Location Awareness in iOS and Android Apps - *James Montemagno*	**W14** Build Data-Centric HTML5 Single Page Applications with Breeze - *Brian Noyes*	**W15** Getting Up & Running with Docker on Microsoft Azure - *Rick Garibay*	**W16** Stop the Waste and Get Out of (Technical) Debt - *Richard Hundhausen*
4:30 PM	5:45 PM	**W17** Swift for .NET Developers - *Jon Flanders*	**W18** Take a Gulp, Make a Grunt, and Call Me Bower - *Adam Tuliper*	**W19** A Pragmatic Reference Architecture for the Internet of Things - *Rick Garibay*	**W20** Professional Scrum Development Using Visual Studio 2015 - *Richard Hundhausen*
7:00 PM	9:00 PM	Rollin' on the River Bat Cruise			

START TIME	END TIME	Visual Studio Live! Day 3: Thursday, June 4, 2015			
8:00 AM	9:15 AM	**TH01** Automated UI Testing for Android and iOS Mobile Apps - *James Montemagno*	**TH02** Busy JavaScript Developer's Guide to ECMAScript 6 - *Ted Neward*	**TH03** SQL 2014 Columnstore Indexes and In-Memory Optimized Tables - *Kevin Goff*	**TH04** What's New in C# 6.0 - *Jason Bock*
9:30 AM	10:45 AM	**TH05** You Too Can Easily Create an Amazing 3D Application with Unity - *Adam Tuliper*	**TH06** Building Web APIs to Support Your HTML/JavaScript Client - *Brian Noyes*	**TH07** SQL Server Reporting Services - Attendees Choose Topics - *Kevin Goff*	**TH08** Managing the .NET Compiler - *Jason Bock*
11:00 AM	12:15 PM	**TH09** Strike Up A Conversation with Cortana on Windows Phone - *Walt Ritscher*	**TH10** To Be Announced	**TH11** Build Real-Time Websites and Apps with SignalR - *Rachel Appel*	**TH12** Busy Developer's Guide to NoSQL - *Ted Neward*
12:15 PM	1:30 PM	Lunch			
1:30 PM	2:45 PM	**TH13** Windows, NUI, and You - *Brian Randell*	**TH14** Hate JavaScript? Try TypeScript - *Ben Hoelting*	**TH15** ASP.NET MVC: All Your Tests Are Belong To Us - *Rachel Appel*	**TH16** Microsoft's .NET is Now Open Source and Cross-Platform. Why It Matters. - *Mark Rosenberg*
3:00 PM	4:15 PM	**TH17** WPF Data Binding in Depth - *Brian Noyes*	**TH18** JavaScript Patterns for the C# Developer - *Ben Hoelting*	**TH19** Building Rich UI with ASP.NET MVC and Bootstrap - *Walt Ritscher*	**TH20** Async and Await Best Practices - *Mark Rosenberg*

Sessions and speakers subject to change.

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

San Francisco AGENDA AT-A-GLANCE

Cloud Computing	Mobile Client	Database and Analytics	Web Development	Visual Studio / .NET	Windows Client

START TIME	END TIME	Visual Studio Live! Pre-Conference Workshops: Monday, March 16, 2015 (Separate entry fee required)		
9:00 AM	6:00 PM	**M01** Workshop: Big Data, Analytics and NoSQL: Everything You Wanted to Learn But Were Afraid to Ask - *Andrew Brust*	**M02** Workshop: Native Mobile App Development for iOS, Android and Windows Using C# - *Marcel de Vries*	**M03** Workshop: ALM and DevOps with the Microsoft Stack - *Brian Randell*
6:45 PM	9:00 PM	Dine-A-Round with Speakers		

START TIME	END TIME	Visual Studio Live! Day 1: Tuesday, June 16, 2015			
8:00 AM	9:00 AM	**KEYNOTE:** The Future of Application Development - Visual Studio 2015 and .NET 2015 - *Jay Schmelzer, Director of Program Management, Visual Studio Team, Microsoft*			
9:15 AM	10:30 AM	**T01** Azure 10-10: Top 10 Azure Announcements in "T-10" Months - *Vishwas Lele*	**T02** AngularJS 101 - *Deborah Kurata*	**T03** UX Design Principle Fundamentals for Non-Designers - *Billy Hollis*	**T04** A Lap Around Visual Studio 2015 - *Robert Green*
10:45 AM	12:00 PM	**T05** Cloud or Not, 10 Reasons Why You Must Know "Web Sites" - *Vishwas Lele*	**T06** AngularJS Forms and Validation - *Deborah Kurata*	**T07** Designing and Building UX for Finding and Visualizing Data in XAML Applications - *Billy Hollis*	**T08** To Be Announced
12:00 PM	1:30 PM	Lunch — Visit Exhibitors			
1:30 PM	2:45 PM	**T09** Building Mobile Cross-Platform Apps with C# and Xamarin - *Nick Landry*	**T10** ASP.NET 5 in all its Glory - *Adam Tuliper*	**T11** Getting Started with Universal Apps for Windows and Windows Phone - *Philip Japikse*	**T12** To Be Announced
3:00 PM	4:15 PM	**T13** Building Mobile Cross-Platform Apps in C# with Azure Mobile Services - *Nick Landry*	**T14** Hack Proofing Your Web Applications - *Adam Tuliper*	**T15** Building Windows 10 LOB Apps - *Robert Green*	**T16** What's New in ALM - *Brian Randell*
4:15 PM	5:30 PM	Welcome Reception			

START TIME	END TIME	Visual Studio Live! Day 2: Wednesday, June 17, 2015			
8:00 AM	9:00 AM	**KEYNOTE:** To Be Announced - *Kris Lankford, Sr. Product Manager, Microsoft*			
9:15 AM	10:30 AM	**W01** iOS Native Development - *Jon Flanders*	**W02** Implementing M-V-VM (Model-View-View Model) for WPF - *Philip Japikse*	**W03** Moving Web Apps to the Cloud - *Eric D. Boyd*	**W04** Stop the Waste and Get Out of (Technical) Debt - *Richard Hundhausen*
10:45 AM	12:00 PM	**W05** Swift for .NET Developers - *Jon Flanders*	**W06** Strike Up a Conversation with Cortana on Windows Phone - *Walt Ritscher*	**W07** Solving Security and Compliance Challenges with Hybrid Clouds - *Eric D. Boyd*	**W08** Best Practices for Using Open Source Software in the Enterprise - *Marcel de Vries*
12:00 PM	1:30 PM	Birds-of-a-Feather Lunch — Visit Exhibitors			
1:30 PM	2:45 PM	**W09** Building Cross Platform UI with Xamarin.Forms - *Walt Ritscher*	**W10** Take a Gulp, Make a Grunt, and Call Me Bower - *Adam Tuliper*	**W11** To Be Announced	**W12** Automated Build, Test & Deploy with TFS, ASP.NET, and SQL Server - *Benjamin Day*
3:00 PM	4:15 PM	**W13** Adding Analytics to Your Mobile Apps on Any Platform with Microsoft Application Insights and Hockeyapp - *Marcel de Vries*	**W14** Securing Angular Apps - *Brian Noyes*	**W15** Getting Up & Running with Docker on Microsoft Azure - Rick Garibay	**W16** Professional Scrum Development Using Visual Studio 2015 - *Richard Hundhausen*
4:30 PM	5:45 PM	**W17** Creating Applications Using Android Studio - *Kevin Ford*	**W18** - Build Data-Centric HTML5 Single Page Applications with Breeze - *Brian Noyes*	**W19** A Pragmatic Reference Architecture for The Internet of Things - *Rick Garibay*	**W20** Load Testing ASP.NET & WebAPI with Visual Studio - *Benjamin Day*
7:00 PM	9:00 PM	Visual Studio Live! Evening Event			

START TIME	END TIME	Visual Studio Live! Day 3: Thursday, June 18, 2015			
8:00 AM	9:15 AM	**TH01** Using Multi-Device Hybrid Apps to Create Cordova Applications - *Kevin Ford*	**TH02** Build Real-Time Websites and Apps with SignalR - *Rachel Appel*	**TH03** Windows, NUI, and You - *Brian Randell*	**TH04** To Git or Not to Git for Enterprise Development - *Benjamin Day*
9:30 AM	10:45 AM	**TH05** Everything You Always Wanted To Know About REST (But Were Afraid To Ask) - *Jon Flanders*	**TH06** Knocking it Out of the Park with Knockout.JS - *Miguel Castro*	**TH07** Building Rich Data Web APIs with ASP.NET OData - *Brian Noyes*	**TH08** What's New in C# 6.0 - *Jason Bock*
11:00 AM	12:15 PM	**TH09** Comparing Performance of Different Mobile Platforms - *Kevin Ford*	**TH10** To Be Announced	**TH11** Power BI 2.0: Analytics in the Cloud and in Excel - *Andrew Brust*	**TH12** Microsoft's .NET is Now Open Source and Cross-Platform. Why it Matters. - *Mark Rosenberg*
12:15 PM	1:30 PM	Lunch			
1:30 PM	2:45 PM	**TH13** Programming WPF with MVVM - Advanced Topics - *Miguel Castro*	**TH14** Busy JavaScript Developer's Guide to ECMAScript 6 - *Ted Neward*	**TH15** Big Data and Hadoop with Azure HDInsight - *Andrew Brust*	**TH16** Managing the .NET Compiler - *Jason Bock*
3:00 PM	4:15 PM	**TH17** Extending XAML To Overcome Pretty Much Any Limitation - *Miguel Castro*	**TH18** ASP.NET MVC: All Your Tests Are Belong To Us - *Rachel Appel*	**TH19** Busy Developer's Guide to NoSQL - *Ted Neward*	**TH20** Async and Await Best Practices - *Mark Rosenberg*

Sessions and speakers subject to change.

Visualize Streaming Data the Easy Way with OData

Louis Ross

The Open Data Protocol (OData) is a RESTful protocol designed to allow powerful query and modification operations on data in a backing store, typically a SQL database. Both resource expressions and queries are formed through the URL of an HTTP request, with results returned in the HTTP response. Sophisticated support for queries that can shape, order, filter and page data requests is built in through a query language. Because OData is an OASIS standard, it's widely implemented and can be consumed across all popular client platforms, such as Web browsers, as well as phones and devices based on iOS, Android and Windows. OData is frequently viewed as a good way to provide a standards-based service that can be consumed across multiple platforms easily. For some starter links, see "Additional References."

In this article, I'll demonstrate why the semantics of OData make it the perfect vehicle for exposing near-real-time time-series streaming data, as well as data from SQL backing stores. (I'll use the term *time-series* to differentiate from *static* backing stores. This avoids the use of *real-time*, which has different meanings in different contexts.)

This article will demonstrate a sample implementation of an industrial automation time-series data streaming capability using an OData service. **Figure 1** shows a sample test client, written in C# using Windows Presentation Foundation (WPF), that connects to the service and uses simple LINQ queries to process the time-series data stream. I'll discuss this in more detail shortly. As I mentioned, many possible clients exist, including browser-based and device apps.

The client connects with an OData service to manage multiple items and subscriptions. The membership of items in each subscription, the subscription update interval and the dynamic monitoring of subscriptions can all be managed through the test client. I'll cover the concepts of items and subscriptions in more detail later. **Figure 2** shows another window of the test client, with time-series data being streamed from one of the subscriptions.

This technique can be extended to any time-series data from devices on the Internet of Things (IoT). As the IoT continues to build out, devices from rain sensors to point-of-sale terminals are becoming available directly over the Internet. The further this process goes, the greater the need for a simple, consistent way to access and manage all of the data. The technique I show here is ideally suited to traverse the Internet, allowing data from IoT devices to be consumed by computers and other devices, making the information accessible and meaningful.

My example is for demonstration purposes only, and is scoped in capability for purposes of clarity: all of the items it exposes are of the same integer type; the "device" providing the data is a

This article discusses:

- Designing the entity model for an OData service
- Implementing a sample OData service using WCF Data Services
- Creating an OData client in Visual Studio
- Writing a simple Windows Presentation Foundation OData test client

Technologies discussed:

OData, WCF Data Services, LINQ, Visual Studio

Code download available at:

msdn.microsoft.com/magazine/msdnmag0415

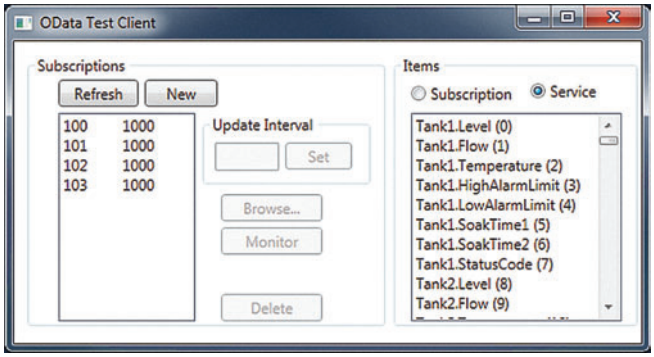


Figure 1 **A Sample OData Test Client Written in C# Using Windows Presentation Foundation**

simulation; and no attempt is made to authenticate a user or provide a user identity and per-user session. All these things would be necessary for a production service, but have been eliminated for illustration purposes.

Design the Entity Model

The first step in defining an OData service is designing the entity model. An OData service is exposed at a specific base URL, such as http://localhost:10001/TimeSeriesData.svc, and each URL segment below this base URL represents a different kind of resource within the entity model. The base URL may change, depending on how the OData service is deployed, but the entity model will dictate a fixed set of resource URL segments under it.

With typical OData services that are backed by a database, resources can be modeled using an automated tool such as ADO.NET Entity Framework. My time-series OData service, on the other hand, exposes resources as in-memory structures, so I generated the entity model by hand. A reasonable set of resources for a time-series data streaming service might be Items, Subscriptions and Samples:

Get All Items: http://localhost:10001/TimeSeriesData.svc/Items should return all the items available in the device for which the service supplies data. Each item should have properties such as id, name, type, state, read-write capabilities, current value and so on.

Get All Subscriptions: http://localhost:10001/TimeSeriesData.svc/Subscriptions should return all the subscriptions that have been created and persisted by previous calls to the service. A subscription is a mechanism to group items into collections. An additional feature of a subscription is that the items grouped into each subscription are sampled together at a specific rate. Each subscription should have properties such as id, subscription interval, its collection of items, and its collection of samples for those items.

Get All Samples: http://localhost:10001/TimeSeriesData.svc/Samples should return all the samples exposed globally by the service. For this article, this query never returns anything, because in real-life scenarios it's a meaningless query. Samples are always associated with a subscription, so should not be queried at the root. However, for WCF Data Services to expose samples as entities, the root query must be valid.

In addition to the three basic entities, I added associations between them. The Subscription resource is associated one-to-many with both Samples and Items.

Get All Samples Within a Subscription: http://localhost:10001/TimeSeriesData.svc/Subscriptions(25)/Samples should return all the samples available for subscription 25. This assumes the client has previously created a subscription entity with SubscriptionId = 25. Each sample should have properties such as id; the id of the item that was sampled; and the value, time and quality of the item when it was sampled.

Get All Items Within a Subscription: http://localhost:10001/TimeSeriesData.svc/Subscriptions(25)/Items should return the collection of items currently associated with subscription 25. This assumes that the client previously created a subscription entity with SubscriptionId = 25 and associated items with it.

Figure 3 shows the entity model of the time-series OData service. Items are physical entities in the device, usually addressable through a string name or numeric id. Items typically contain a value exposed by the device, such as temperature or pressure, along with metadata such as the timestamp of the most recent sample and quality of the measurement. As **Figure 3** shows, they're exposed as OData entities by their numeric ItemId.

As an entity, the Subscription is a fiction invented to group subsets of device items together, and to collect samples from that subset at a fixed period. Because subscriptions don't really exist, there must be some mechanism in the service to generate the entities in such a way that they appear to be backed by a store.

Sample entities are even more ephemeral than Subscriptions. The sample entity is invented to capture the value and metadata of a single item at a specific time. When a subscription is operating, it typically accumulates Sample entities as needed to capture the recent samples of items associated with that subscription. The fiction with Sample entities is that a vast table of entities exists to fulfill OData requests. In reality, Sample entities are deleted from storage as fast as they are returned to the client. Because clients are typically not interested in obtaining the same sample more than once, this is not an issue.

Figure 2 **Time-Series Data Streaming from a Subscription in the OData Service**

Dealing with these in-memory abstractions in a manner that removes statefulness from the service runtime is the crux of a solution that allows an OData service to scale—sequential calls from a client shouldn't depend on the service remembering details about the processing of subscriptions or samples in previous calls.

This means abstract entities such as Subscriptions and Samples must never be remembered as state in the OData service. Instead, they must be considered part of the backing service or device. In my example OData service, I implemented a simulation layer to remove any dependency on real hardware. The entities are provided by the simulated hardware, and simply exposed by the OData service. In particular, Item and Subscription entities are persisted by the simulation layer to an XML file.

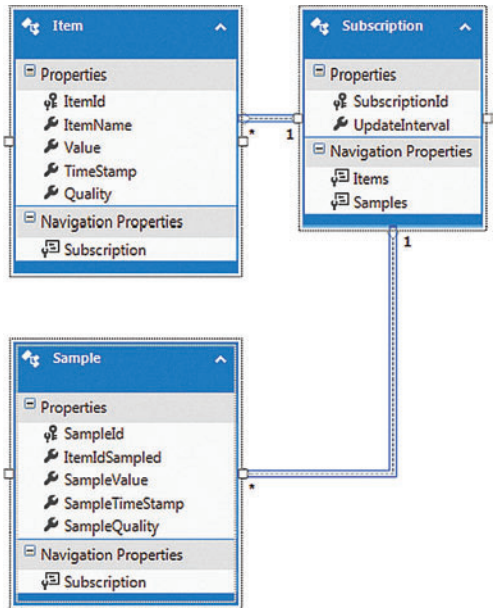


Figure 3 **The Entity Model Exposed by the Time-Series OData Service**

The Sample entity is a special case. Samples are virtual in the sense that no Sample entity exists in the simulated hardware, but Samples are also ephemeral. Typical time-series clients—as opposed to historical clients—are interested only in the most recent Samples. Once those Samples are read, they can be discarded in the service. In my sample code, I provide each Sample entity with a unique Id, and maintain the fiction that the OData client might at any time ask for really old Samples. However, older Samples are forgotten by the simulated hardware layer, so any client that actually did this would be disappointed.

Implement the Sample OData Service Using WCF Data Services

Now that the entity model is defined, the next step is to implement an OData service. The ASP.NET Web API framework is a powerful modern choice, but I fell back on the older WCF Data Services for its simpler implementation of the basics.

The class TimeSeriesDataService exposes the service by deriving from the template class DataService:

```
public class TimeSeriesDataService : DataService<TimeSeriesEntities>
{
  public static void InitializeService(DataServiceConfiguration config)
  {
    // Initialize the service here...
  }
}
```

The template parameter of the DataService base class specifies the class that defines the entity model for this service, TimeSeriesEntities. To allow for the queries shown in the earlier design section, the TimeSeriesEntities class exposes the Sample, Item and Subscription entities as queryable collections, as shown in **Figure 4**.

By providing public properties that return IQueryable<T> collections, you enable the WCF Data Services framework to process all the powerful OData queries that can be expressed as HTTP URLs. Your code provides the queryable collections, and the framework performs the actual filtering, sorting, pattern matching and so forth.

Make the OData Service Writable

To allow entities to be modified in the service, you have to add a little more code. This code will be called by the WCF Data Services framework in response to modification commands sent by the HTTP client using POST, PUT or DELETE verbs. A typical time-series data service allows the client to modify the state of the service in various ways: write the values of items; create and delete subscriptions; add and remove items from subscriptions; and more.

When using WCF Data Services, this modification ability is provided by implementing the IUpdatable interface. When a client sends an HTTP POST with a description of an entity to create, the Data Services framework handles the request, passing requests to the 12 IUpdatable interface methods, as needed.

My first service modification code allows the creation and deletion of subscriptions. You'll need to implement other IUpdatable interface methods to support other modification capabilities, such as writing a value to an item.

I enable creation and deletion of subscriptions by implementing the CreateResource and DeleteResource IUpdatable methods, as shown in **Figure 5**.

The WCF Data Services framework manages the bulk of the POST, PUT, and DELETE client calls, and simply needs my code to deal with

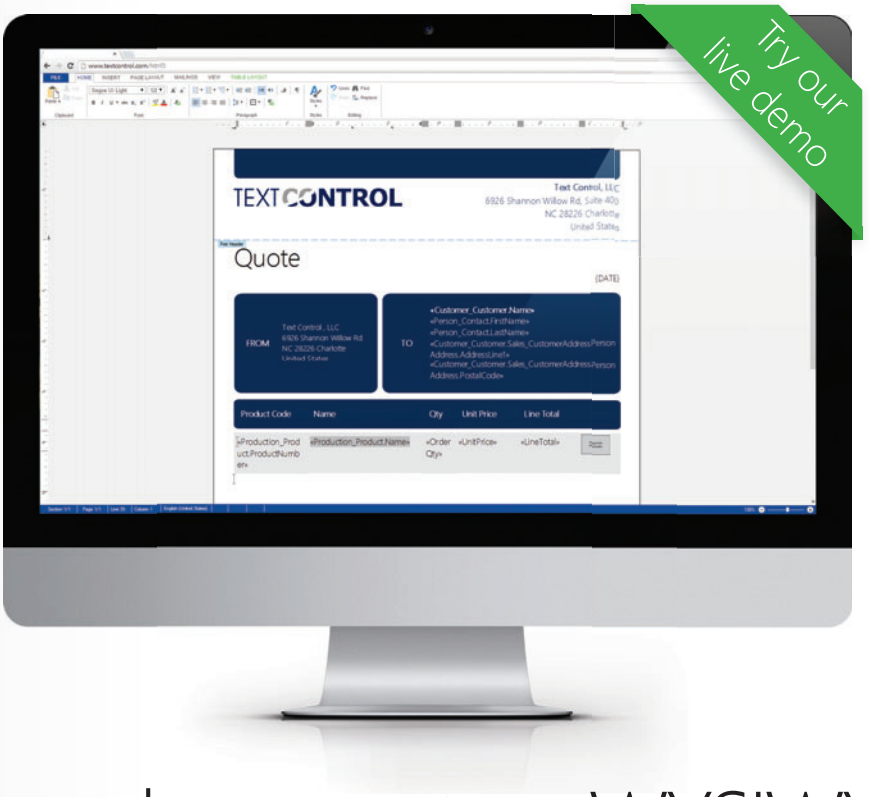
Figure 4 **The Entity Model in Code**

```
public partial class TimeSeriesEntities
{
  public IQueryable<Sample> Samples
  {
    get { return (new List<Sample>()).AsQueryable<Sample>(); }
  }

  public IQueryable<Item> Items
  {
    get
    {
      return (from item in ItemRegistry.Items
              select new Item(item.Id,
                              item.Name,
                              item.Value,
                              item.Time,
                              (int)item.Quality))
               .AsQueryable<Item>();
    }
  }

  public IQueryable<Subscription> Subscriptions
  {
    get
    {
      return (from subscription in SubscriptionRegistry.Subscriptions
              select new Subscription() { SubscriptionId =
              subscription.Id })
             .AsQueryable<Subscription>();
    }
  }
}
```

Finally!



Cross-browser, true WYSIWYG document editing for ASP.NET

- Edit and create MS Word documents
- Create and modify Adobe® PDF documents
- Create reports and mail merge templates
- Integrate with Microsoft® Visual Studio

Try it online and download your 30-day trial version today:
www.textcontrol.com/html5

Visual Studio Partner

HTML 5

TEXT CONTROL

Figure 5 **Creating and Deleting Subscriptions**

```
public partial class TimeSeriesEntities : IUpdatable
{
  object IUpdatable.CreateResource(string containerName,
    string fullTypeName)
  {
    Type t = Type.GetType(fullTypeName, true);
    return Activator.CreateInstance(t);
  }

  void IUpdatable.DeleteResource(object targetResource)
  {
    MethodInfo deleteMethod =
      targetResource.GetType().GetMethod("Delete");
    if (deleteMethod != null && deleteMethod.IsPublic)
    {
      deleteMethod.Invoke(targetResource, null);
    }
  }
}
```

any types specific to my internal implementation. You can choose to implement static factory methods, or use reflection to create new instances, as shown in **Figure 5**. My choice of mechanisms to delete resource objects relies on an implementation detail—my resource objects all expose a method called Delete, which can be called from DeleteResource. You can choose other implementations as desired.

OData Client Made Easy with LINQ

No matter how a time-series OData service is implemented and exposed, its power lies in what a consuming client can do with it. This is especially true with the Microsoft .NET Framework, because LINQ has a set of extensions that directly interface to OData services, passing as much of a query to the service as possible. The use of LINQ makes it easier for client code to manage complex queries, including filters, sorts and others with relative ease.

Create an OData Client in Visual Studio

With my sample OData service running, I started Visual Studio and pointed the Add Service Reference tool to the base URL of the OData service. The tool examined the OData metadata and created a class derived from System.Data.Services.Client.DataServiceContext, and called it TimeSeriesEntities. This class is customized for the OData service, and includes properties that return typed DataServiceQuery<TElement> objects, customized for each of the IQueryable<TElement> collections in the WCF Data Services code:

```
public DataServiceQuery<Sample> Samples { get; }
public DataServiceQuery<Item> Items { get; }
public DataServiceQuery<Subscription> Subscriptions { get; }
```

Each of these DataServiceQuery<TElement> instances implements IEnumerable, so each is a target for LINQ queries. Where possible, LINQ queries formed against DataServiceQuery<TElement> objects will translate into OData queries, allowing much of the filtering, sorting and pattern matching to occur at the service.

In the examples that follow, I'll be using the service base URL http://localhost:10001/TimeSeriesData.svc, contained in the variable dataAddress. For all the query examples that follow, a TimeSeriesEntities object named dataContext is assumed to exist, and to have been initialized with the service base URL:

```
TimeSeriesDataFeed.TimeSeriesEntities dataContext =
  new TimeSeriesDataFeed.TimeSeriesEntities(dataAddress);
```

As a starter example, the OData URL http://localhost:10001/TimeSeriesData.svc/Items is generated by the following LINQ query:

```
var serviceQuery = from item in dataContext.Items
                   select item;
```

The returned LINQ query can be executed with:

```
serverItemsList = serviceQuery.ToList();
```

sending the URL and receiving and parsing the OData response, resulting in an enumeration of all Item entity objects available at the root service level.

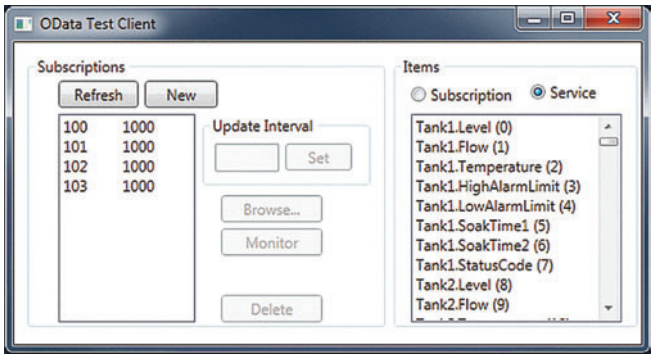


Figure 6 **The OData Test Client Main Window**

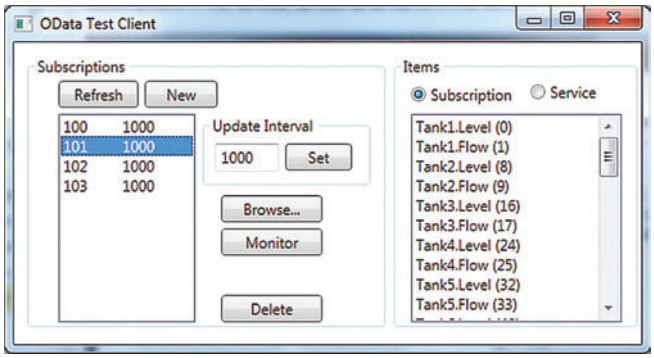


Figure 7 **A Subscription Is Selected**

Figure 8 **Monitoring a Single Subscription**

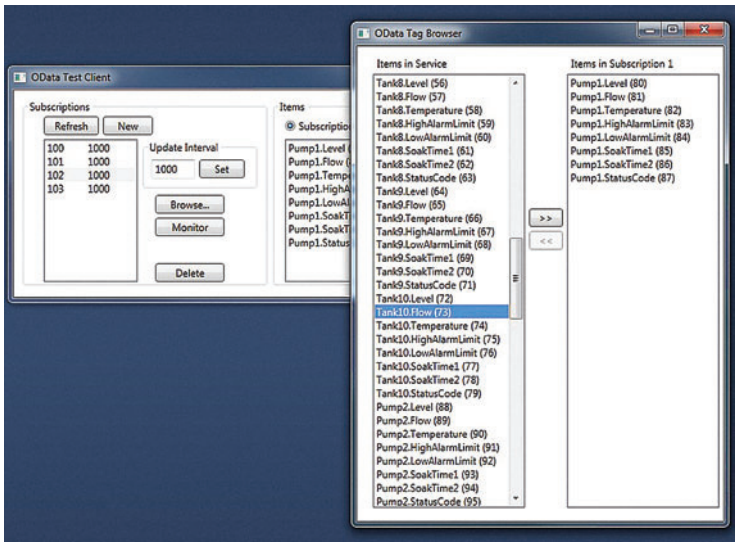


Figure 9 **Browsing Items to be Included in a Subscription**

Note that this query is similar to browsing all items in the device namespace. In other protocols, the ability to browse like this is an added feature, and much effort is expended to define and implement the capability. With OData, it comes for free!

A more nuanced query might be to enumerate all items currently associated with a specific subscription, for example, the subscription with SubscriptionId = 25. The LINQ query:

```
var subscriptionQuery =
  (from subscription in dataContext.Subscriptions.Expand("Items")
  where subscription.SubscriptionId == 25
  select subscription).FirstOrDefault();
```

generates the OData URL:

```
http://localhost:10001/TimeSeriesData.svc/Subscriptions(25)/?$expand=Items
```

The call to FirstOrDefault dereferences the returned LINQ query and executes it, sending the URL and receiving and parsing the response. This time, the result is a single Subscription entity object with all Item entity objects associated with the subscription. The subscription items can be enumerated with:

```
serverItemsList = serverSubscription.Items.ToList();
```

Client code pulls published samples from a subscription using a LINQ query like this:

```
var subscriptionWithSamplesQuery =
  (from subscription in dataContext.Subscriptions.Expand("Samples")
  where subscription.SubscriptionId == 25
  select subscription).FirstOrDefault();
```

which generates the OData URL:

```
http://localhost:10001/TimeSeriesData.svc/Subscriptions(25)/?$expand=Samples
```

In this case, Sample entities associated with the Subscription entity contain sampled values for many Item entities, possibly one or more sample for each item associated with the subscription. As long as I'm using LINQ, I'll group samples by item with a local LINQ query operating on the IEnumerable collection returned by the previous LINQ query:

```
var samplesInSubs = from sample in subscriptionWithSamplesQuery.Samples
                    group sample by sample.ItemIdSampled into sampleSubs
                    select sampleSubs;
```

This example is the basic-course use case of time-series data acquisition. It demonstrates how naturally LINQ queries that generate OData transactions can segue into in-memory LINQ queries that manipulate local data. At this point, samplesInSubs is an IEnumerable of IEnumerable, a collection of groups of samples, grouped by ItemId. It's natural for the client to process the data in this form.

These examples of OData queries don't even begin to touch the expressiveness of OData. Much more filtering, sorting and sampling is possible. See"Additional References" for links to more examples.

A Simple WPF OData Test Client

Putting everything together, I wrote a small WPF application to showcase the concept of time-series process data from an OData service. The OData Test Client shown in **Figures 6-10** uses the LINQ queries I explored earlier, plus a few others, to read data from and write data to a sample OData service.

When the main screen first comes up (see **Figure 6**), it makes several OData queries, allowing it to fill in the list boxes with all subscriptions in the service, as well as names and Ids of all items in the service.

In **Figure 7**, you see that selecting a subscription in the left list box reads the item names and Ids of the items in that subscription into the right list box, which allows you to read and set the update interval of that subscription, and enables several command buttons for that subscription, which I'll describe as I go.

No discussion of time-series streaming data is complete without a strip-chart recorder, so I found the excellent Dynamic Data Display project at CodePlex, and used it to build a multi-pen dynamic display. (Note that the Dynamic Data Display for Silverlight library is a Microsoft Research product and its license states that the library is for non-commercial use only.) Data for all items in a single subscription roll as expected from right to left. You get a subscription monitor dialog like the one in **Figure 8** every time you select a subscription and press Monitor.

In addition to monitoring time-series data from many items, the same OData queries can be used to perform item browsing, as shown in **Figure 9**. All the items from either a single subscription or the service as a whole are displayed in the list box on the right, but browsing also means choosing which items to include in a subscription. To get a browsing window, select a subscription

Additional References

OData Web Site: odata.org

OASIS OData Standard: bit.ly/163s1gZ

OData Query Language: See the canonical description at bit.ly/15PVBXv or Microsoft's take, including limitations of using LINQ to generate the queries, at bit.ly/1zZYMqq.

OData, CKAN AND Microsoft Azure: How OData typically provides access to large data sets (bit.ly/1LrzmYp).

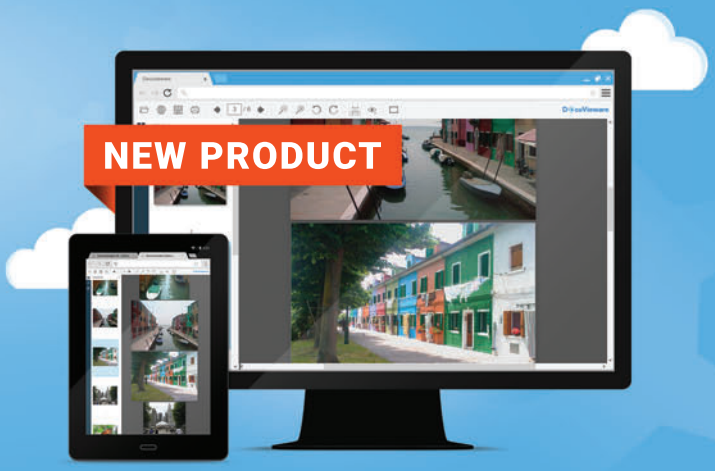
LINQ Queries Mapped to OData Queries: There are other third-party extensions that extend LINQ beyond the WCF Data Services Client Library. The basic WCF LINQ queries, and how they map to OData queries, are detailed at bit.ly/1zf8Gp7.

Dynamic Data Display: I found this really excellent set of controls for dynamic data display at bit.ly/1Cl73B2. Using it for strip-chart recorders barely scratches the surface of its capabilities.

DocuVieware

Universal HTML5 Viewer and Document Management Kit

View, annotate and manage any document, on any device, on any browser through a fully customizable zero-footprint HTML5 Control. Supports 90+ file formats, including PDF, TIFF and SVG.



Why **DocuVieware**?

Cross-platform, any browser, **zero-footprint** solution (no client-side install).

Super-**easy integration** within existing web applications.

Fast and **crystal-clear rendering** of documents and annotations.

Fully **customizable UI** look and feel, including convenient **snap-ins**.

Mobile devices optimization & **responsive** design.

Built-in **annotations**, **thumbnails**, **bookmarks** and **text search** tools.

Works in all modern browsers ...

IE9+, Chrome, Chrome for Android, Firefox, Firefox for Android, Opera, Safari 5+, Mobile Safari.

... so it works on all devices.

PC, Mac, tablets and smartphones.

Powered by GdPicture.NET 11

www.docuvieware.com

on the left and press the Browse button. The changes made to that subscription by moving items left (out of the subscription) or right (into the subscription) are updated in the service and persisted each time a service call is made, so no Save button is needed.

Finally, many subscriptions can be monitored at once, as shown in **Figure 10**.

Challenges and Limitations of OData for Time-Series Streaming Data

Although my company specializes in high-density, high-throughput time-series streaming services and clients, no attempt has been made (yet) to quantify the performance of OData for this use. As I compare my OData approach against other approaches common in my industry, I see potential for some downsides:

- No data acquisition mechanism can be safely used in today's networking environment without security, including mandatory user authentication and optional encryption. The obvious way to provide these features is to shift from HTTP to HTTPS. The additional burden of public key infrastructure (PKI) administration and loading of both CPU and bandwidth resources may make this approach unworkable for some applications.
- The approach detailed in this article requires that the service persist some state, such as the definition of multiple subscriptions. When combined with user authentication, this means some state must be persisted at the service side for each user. For scalability reasons, care must be taken to ensure any required state information will be incorporated into the entity model, and not be implemented as part of the service itself. As long as statelessness in the service itself is carefully preserved, the service should scale linearly with added clients, and multi-tenant cloud hosting and server farms should be possible.
- Caching responses for clients that make rapid repeated service requests could be a performance enhancement. This issue comes in two flavors: the Subscriptions(x)/Samples entity set and everything else. Samples from a specific subscription are likely to be the most common request by far, because that's how streaming data is acquired. This works well with the service design, because subscription samples are by definition always in-memory. If other request types occur with enough frequency to be a problem, additional caching might need to be considered.
- OData wire performance may limit its effective item density or throughput. At my company, we haven't yet made measurements to determine this, and it's only a possibility. When compared to SOAP-based transports, the JSON payload from OData may compare favorably to SOAP envelope overhead.
- OData and other REST-based services provide no mechanism to report by exception, or to push notifications of important events to the client. All data transfers must be requested by the client, so only a pull model is possible. This limitation is comparable to most other protocols used by the industry that are capable of traversing the Internet. For example, OPC-UA is an industry-standard SOAP-based protocol, and requires notifications to be pumped using a publish-request loop driven by the client. A valuable exploration for future development of this technique would be to explore the use of server-side push technologies such as SignalR.

From the examples in this article, it seems clear that OData is a feasible approach with several benefits:

- Client code is easier to manage and deploy.
- Multiple client platforms—from phones to desktops—can consume the same data.
- OData is a viable approach for densities of a few dozen to a few hundred items, and throughputs up to a few hundred samples per second, but is untested for high performance.

In my next article, I'll follow up with an exploration of how to consume "data in flight" from OData services with Reactive Extensions (Rx). ■

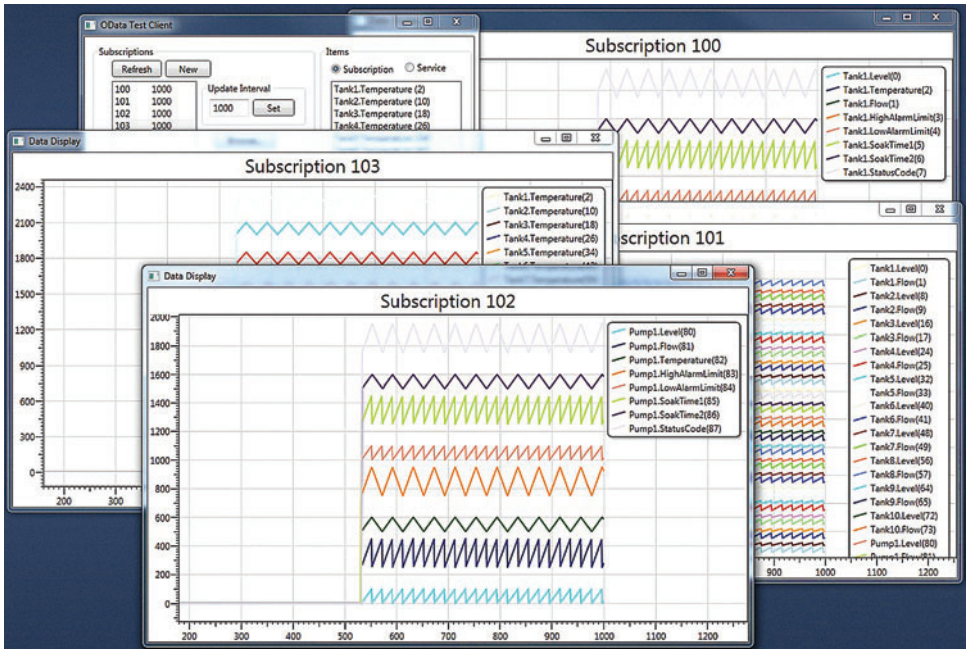


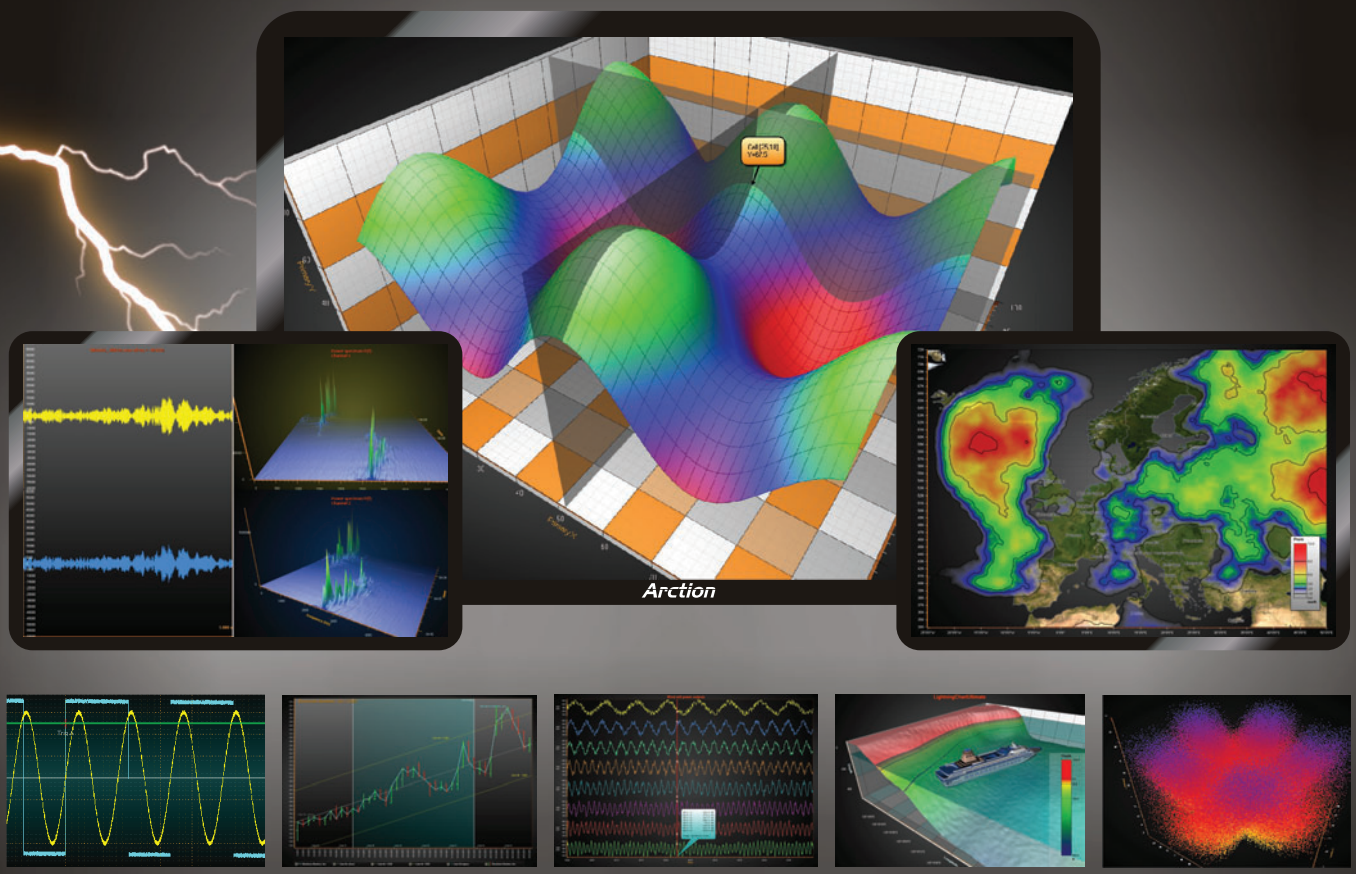
Figure 10 **Monitoring Four Subscriptions at Once**

Louis Ross *is a multi-technology architect for Wonderware by Schneider Electric, where he has been developing device-integration technologies for 17 years. Previously, he freelanced as a designer of embedded controls, specializing in hardware and firmware design.*

Thanks to the following technical experts for reviewing this article and providing valuable feedback: *Ming Fong (Schneider Electric) and Jason Young (Microsoft)*

The FASTEST rendering Data Visualization components for WPF and WinForms...

LightningChart



HEAVY-DUTY DATA VISUALIZATION TOOLS FOR SCIENCE, ENGINEERING AND TRADING

WPF charts performance comparison

Opening large dataset	LightningChart is up to 977,000 % faster
Real-time monitoring	LightningChart is up to 2,700,000 % faster

WinForms charts performance comparison

Opening large dataset	LightningChart is up to 37,000 % faster
Real-time monitoring	LightningChart is up to 2,300,000 % faster

Results compared to average of other chart controls. See details at www.LightningChart.com/benchmark. LightningChart results apply for Ultimate edition.

- Entirely DirectX GPU accelerated
- Superior 2D and 3D rendering performance
- Optimized for real-time data monitoring
- Touch-enabled operations
- Supports gigantic data sets
- On-line and off-line maps
- Great customer support
- Compatible with Visual Studio 2005...2013
- Now with Audio I/O components

Download a free 30-day evaluation from

www.LightningChart.com

Arction

2D Drawing Techniques and Libraries for Web Games

Michael Oneppo

For a long time, there was really only one way to make interactive Web games: Flash. Like it or not, Flash had a fast drawing system. Everyone used it to create animations, point-and-click adventures and all sorts of other experiences.

When browsers aligned on Web standards with HTML5, there was a veritable explosion of options for developing fast, high-quality graphics—without needing plug-ins. This article will present a small sample of drawing methods, as well as the underlying technologies and some libraries for making them easier to use. I won't be covering libraries specifically meant for games. There are so many of those I'm going to save that discussion for another article.

Drawing Standards

With the onset of HTML5, three common ways for drawing in 2D emerged: the Document Object Model (DOM), the canvas and the Scalable Vector Graphics (SVG) format. Before jumping into the survey of libraries that use these technologies, I'll review how each works to better understand the tradeoffs of each method.

Not surprisingly, the most basic way to draw graphics in HTML is indeed with HTML. By creating a number of image or background elements and using a library like jQuery, you can quickly make sprites you can move around without redrawing the scene. The browser will do it for you. This kind of structure is often called a scene graph. In the case of HTML, the scene graph is the DOM. Because you'll use CSS to style your sprites, you can also use CSS transitions and animations to add some smooth motion to your scene.

The key issue with this method is that it's dependent on the DOM renderer. This can slow things down when you have a complex scene. I wouldn't recommend using more than a few hundred elements. So anything more complex than a match-three or a platform game might have performance issues. And a sudden increase in the number of elements, like in a particle system, can cause hiccups in the animation.

Another issue with this method is you need to use CSS to style elements. Depending on how you write CSS, it can be decently fast or pretty slow. Finally, writing code targeted for HTML can be difficult to move to a different system, like native C++. This is important if you want to port your game to something like a console. Here's a summary of the pros:

- Builds on the basic structure of a Web page
- jQuery and other libraries make it easy to move things around
- Sprites are relatively easy to set up
- Built-in animation system with CSS transitions and animations

And a summary of the cons:

- Many small elements can slow you down
- Need to use CSS to style elements
- No vector images
- Can be difficult to port to other platforms

This article discusses:

- Graphics standards for 2D drawing
- How to incorporate libraries to simplify the process
- Pros and cons of different methods

Technologies discussed:

HTML5, Visual Studio Pro 2013, Visual Studio Community, ASP.NET

HTML5 Canvas

The canvas element addresses a lot of the cons. It provides an immediate-mode rendering environment—a flat swath of pixels. You tell it what to draw in JavaScript and it draws immediately. Because it's converting your drawing commands to pixels, you can quickly pile up a long list of drawing commands without bogging down the system. You can draw geometry, text, images, gradients and other elements. To read more about using canvas for games, check out David Catuhe's article at bit.ly/1fquBuo.

So what's the downside? Because the canvas forgets what it drew the moment it's done, you have to redraw the scene yourself each time you want it to change. And if you want to modify a shape in a complex way, like bending or animating, you have to do the calculations and redraw the item. This means you need to maintain a lot of data about your scene in your own data structures. That's not really a huge deal, considering there are libraries that make this easier. If you really want to do something customized, be aware that the canvas doesn't retain information for you. Finally, canvas doesn't include animations. You have to draw your scene in successive steps to make a smooth animation. Here's a summary of the pros:

- Draws directly—scenes can be more complex
- Supports lots of different visual elements

And a summary of the cons:

- No inherent memory of the scene; you need to build it yourself
- Complex transforms and animations have to be done manually
- No animation system

SVG: Scalable Vector Graphics

As an XML-based markup for describing 2D visuals, SVG is similar to HTML. The key difference is SVG is meant for drawing, while HTML is primarily designed for text and layout. As such, SVG has some powerful drawing capabilities such as smooth shapes, complex animations, deformations and even image filters like blurring. Like HTML, SVG has a scene graph structure, so you can peruse SVG elements, add shapes, change their properties and not worry about redrawing everything. The browser will do it for you. The video, "Working with SVG in HTML5," from Channel 9 (bit.ly/1DEAWmh) explains more.

Like HTML, complex scenes can bog down SVG. SVG can handle some complexity, but it can't quite match the complexity afforded by using canvas. Furthermore, the tools for manipulating SVG can be complex, although there are other tools to simplify the process. Here's a summary of the pros:

- Many drawing options like curved surfaces and complex shapes
- Structured with no need to redraw

And a summary of the cons:

- Complexity can bog it down
- Difficult to manipulate

> Because the canvas forgets what it drew the moment it's done, you have to redraw the scene yourself each time you want it to change.

2D Drawing Libraries

Now that you know about the standards available to draw on the Web, I'll take a look at some libraries that can make your drawing and animation easier. It's worth noting that rarely do you draw without doing something else with the drawing. For example, you often need graphics to react to input. Libraries help make common tasks associated with drawing easier.

KineticJS Want a scene graph for the canvas? KineticJS is an extremely powerful canvas library that starts with a scene graph and adds more functionality. At the baseline, KineticJS lets you define layers in the canvas that contain shapes to draw. For example, **Figure 1** shows how to draw a simple red circle using KineticJS.

You'll need to call the final line of **Figure 1** every time you want the scene to redraw. KineticJS will do the rest by remembering the layout of the scene and ensuring everything is drawn correctly.

There are some interesting things in KineticJS that make it quite powerful. For example, the fill property for an object can be quite a number of things, including a gradient:

```
fill: {
  start: {x: 0, y: 0},
  end: {x: 0, y: 200},
  colorStops: [0, '#FF0000', 1, '#00FF00']
},
```

Or an image:

```
// The "Image" object is built into JavaScript and
// Kinetic knows how to use it
fillPatternImage: new Image('path/to/an/awesome/image.png'),
```

KineticJS has an animation system, as well, which lets you move things around by creating an Animation object or by using a Tween object to transition properties on the shapes in your scene. **Figure 2** shows both types of animations.

KineticJS is powerful and widely used, especially for games. Check out the code, samples and documentation at kineticjs.com.

Paper.js Paper.js provides more than just a library for simplifying drawing to the canvas. It provides a slightly modified version of JavaScript

Figure 1 **Drawing a Circle with KineticJS**

```
// Points to a canvas element in your HTML with id "myCanvas"
var myCanvas = $('#myCanvas');
var stage = new Kinetic.Stage({
  // get(0) returns the first element found by jQuery,
  // which should be the only canvas element
  container: myCanvas.get(0),
    width: 800,
    height: 500
  });

var myLayer = new Kinetic.Layer({id: "myLayer"});
stage.add(myLayer);
var circle = new Kinetic.Ellipse({
  // Set the position of the circle
  x: 100,
  y: 100,

  // Set the size of the circle
  radius: {x: 200, y: 200},

  // Set the color to red
  fill: '#FF0000'
});

myLayer.add(circle);
stage.draw();
```

called PaperScript to simplify common drawing tasks. When including PaperScript in your project, link to it just like you would a regular script, just with a different type of code:

```
<script type="text/paperscript" src="mypaperscript.js">
```

This lets Paper.js interpret the code slightly differently. There are really only two pieces to this. First, PaperScript has two built-in objects called Point and Size. PaperScript includes these objects for common use in its functions and provides the ability to directly add, subtract and multiply these types. For example, to move an object about in PaperScript, you could do this:

> Libraries help make common tasks associated with drawing easier.

```
var offset = new Point(10, 10);

var myCircle = new Path.Circle({
  center: new Point(300, 300),
  radius: 60
});

// Direct addition of Point objects!
myCircle.position += offset;
```

The second thing Paper.js interprets differently is responding to events. Consider that you write the following code in JavaScript:

```
function onMouseDown(event) {
  alert("Hello!");
}
```

This will do nothing because the function isn't bound to any element's events. However, writing the same code in PaperScript, Paper.js will automatically detect this function and bind it to the mouse down event. Learn more about this at paperjs.org.

Fabric.js Fabric.js is a feature-packed canvas library, with the ability to mix a number of advanced effects and shapes into a Web page without a lot of code. Some notable features include image filters such as background removal, custom classes for making your own compound objects and "free drawing" support where you can just draw on the canvas in a number of styles. Fabric.js is similar to KineticJS in that it has a scene graph, except with a more concise structure, which some people prefer. For instance, you don't need to ever redraw the scene:

```
var canvas = new fabric.Canvas('myCanvas');
var circle = new fabric.Circle({
  radius: 200,
    fill: '#FF0000',
    left: 100,
    top: 100
});

// The circle will become immediately visible
canvas.add(circle);
```

That's not a huge difference, but Fabric.js provides fine-grained rendering controls that mix automatic redraw and manual redraw. To give an example, scaling up a circle in Fabric.js looks like this:

```
circle.animate(
  // Property to animate
  'scale',
  // Amount to change it to
  3,
  {
    // Time to animate in milliseconds
    duration: 3000,
    // What's this?
    onChange: canvas.renderAll.bind(canvas)
  });
```

When animating something in Fabric.js, you have to tell it what to do when it changes a value. For the most part, you want it to redraw the scene. That's what canvas.renderAll.bind(canvas) references. That code returns a function that will render the whole scene. If you're animating a lot of objects in this way, though, the scene would be unnecessarily redrawn once for each object. Instead, you can suppress redrawing the whole scene, and redraw the animations yourself. **Figure 3** demonstrates this approach.

Fabric.js provides a lot of customization, so you can optimize your drawing only when you need to. For some, that can be difficult to handle. For many complex games, though, this can be a critical feature. Check out more at fabricjs.com.

Raphaël Raphaël is a useful SVG library that eliminates most of the complexity in dealing with SVG. Raphaël uses SVG when available. When it isn't, Raphaël implements SVG in JavaScript using whatever technologies are available in the browser. Every graphical object created in Raphaël is also a DOM object, with all the capabilities DOM objects enjoy, such as binding event handlers and jQuery access. Raphaël also has an animation system that lets you define animations independent of the objects drawn, enabling heavy reuse:

```
var raphael = Raphael(0, 0, 800, 600);

var circle = raphael.circle(100, 100, 200);
circle.attr("fill", "red");
circle.animate({r: 600}, 3000);

// Or make a custom animation
var myAnimation = Raphael.animation(
  {r: 600},
  3000);
circle.animate(myAnimation);
```

In this code, instead of drawing a circle, Raphaël will place an SVG document on the page with a circle element. Oddly, Raphaël doesn't natively support loading SVG files. Raphaël does have a rich community, so there's a plug-in for that available at bit.ly/1AX9n7q.

Snap.svg Snap.svg looks quite a bit like Raphaël:

```
var snap = Snap("#myCanvas"); // Add an SVG area to the myCanvas element
var circle = snap.circle(100, 100, 200);
circle.attr("fill", "#FF0000");
circle.animate({r: 600}, 1000);
```

Figure 2 **Animations Using KineticJS**

```
// Slowly move the circle to the right forever
var myAnimation = new Kinetic.Animation(
  function(frame) {
    circle.setX(myCircle.getX() + 1);
  },
  myLayer);

// The animation can be started and stopped whenever
myAnimation.start();
// Increase the size of the circle by 3x over 3 seconds
var myTween = new Kinetic.Tween({
  node: circle,
  duration: 3,
  scaleX: 3.0,
  scaleY: 3.0
});

// You also have to initiate tweens
myTween.play();
```

One of the key differences is Snap.svg includes seamless SVG importing:

```
Snap.load("myAwesomeSVG.svg");
```

The second key difference is Snap.svg provides powerful built-in tools to search and edit the SVG structure in-place, if you know the structure of the SVG with which you're working. For example, imagine you wanted to make all groups ("g" tags) in your SVG invisible. After SVG loads, you have to add this functionality in a callback to the load method:

```
Snap.load("myAwesomeSVG.svg", function(mySVG) {
  mySVG.select("g").attr("opacity", 0);
});
```

The select method works a lot like the jQuery "$" selector, and it's quite powerful. Check out Snap.svg at snapsvg.io.

A Little More: p5.js

A lot of these libraries provide a little extra for common tasks. This creates a spectrum of technologies to address a wide range of applications, from simple drawing to interactive media and complex gaming experiences. What else is out there in the middle of the spectrum—something more than simple drawing solutions, but not quite a full game engine?

> There are clearly pros and cons to canvas and SVG.

One project worth noting is p5.js, which is built from the popular Processing programming language (see processing.org). This JavaScript library provides an interactive media environment by implementing Processing in the browser. p5.js consolidates the most common tasks into a set of functions you have to define to respond to events in the system, like redrawing the scene or mouse input. It's a lot like Paper.js, but with multimedia libraries, as well. Here's an example, which demonstrates how this approach results in more concise graphics code:

```
float size = 20;
function setup() {
  createCanvas(600, 600);
}

function draw() {
  ellipse(300, 300, size, size);
  size = size + .1;
}
```

This program makes a circle that grows in size until it fills the screen. Check out p5.js at p5js.org.

So What to Use?

There are clearly pros and cons to canvas and SVG. There are also libraries that significantly reduce many of the downsides to either approach. So what should you use? I generally wouldn't recommend using vanilla HTML. Chances are that a modern game will exceed the graphical complexity it can support. So that brings us to a choice between SVG and the canvas, which is difficult.

For highly distinctive game genres, the answer gets a little easier. If you're building a game with hundreds of thousands of particles, you'll want to use the canvas. If you're building a point-and-click adventure game with a comic book style, you might want to consider SVG.

Figure 3 **Tighter Redraw Control in Fabric.js**

```
var needRedraw = true;

// Do things like this a lot, say hundreds of times
circle.animate(
  'scale',
  3,
  {
    duration: 3000,

    // This function will be called when the animation is complete
    onComplete: function() {
      needRedraw = false;
    }
  });

// This function will redraw the whole scene, and schedule the
// next redraw only if there are animations going
function drawAnimations() {
  canvas.renderAll();
  if (needRedraw) {
    requestAnimationFrame(drawAnimations);
  }
}

// Now draw the scene to show the animations
requestAnimationFrame(drawAnimations);
```

For most games, it doesn't come down to performance, as many would have you believe. You can spend hours belaboring which library to use. In the end, though, that's time you could be writing the game.

My recommendation is to make a selection based on your art assets. If you're making your character animations in Adobe Illustrator or Inkscape, why convert each frame of your animations down to pixels? Use the vector art natively. Don't waste all that work by cramming your art into the canvas.

Conversely, if your art is mostly pixel-based, or you're going to be generating complex effects on a pixel-by-pixel basis, the canvas is a perfect option.

One More Option

If you're looking for the best performance possible, and you're willing to deal with a little more complexity to make it happen, I strongly recommend you consider Pixi.js. Unlike anything else I've shown you in this article, Pixi.js uses WebGL for 2D rendering. This provides some major performance improvements.

The API isn't quite as straightforward as the others described here, but it's not a huge difference. Furthermore, WebGL isn't supported on as many browsers as the other technologies. So on older systems Pixi.js has no performance advantage. Either way, make your choice and enjoy the process. ■

Michael Oneppo *is a creative technologist and former program manager at Microsoft on the Direct3D team. His recent endeavors include working as CTO at the technology nonprofit Library for All and exploring a master's degree at the NYU Interactive Telecommunications Program.*

Thanks to the following Microsoft technical expert for reviewing this article: *Shai Hinitz*

NAVIGATE THE .NET HIGHWAY

DEVELOPMENT TRACKS INCLUDE:

- Visual Studio/.NET
- JavaScript/HTML5 Client
- ASP.NET
- Mobile Client
- Windows Client
- Database and Analytics
- Cloud Computing
- SharePoint/Office

REGISTER NOW AND SAVE $400!

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

Use promo code REDAPR2

vslive.com/redmond

Join us on the Ultimate Code Trip in 2015!

LAS VEGAS Code Trip	AUSTIN Code Trip	SAN FRANCISCO Code Trip	REDMOND Code Trip	NEW YORK Code Trip	ORLANDO Code Trip
March 16-20	June 1-4	June 15-18	August 10-14	Sept. 28- Oct. 1	Nov. 16-20

CONNECT WITH VISUAL STUDIO LIVE!

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

vslive.com/redmond

Authoring Desired State Configuration Custom Resources

Ritesh Modi

Desired State Configuration (DSC) is a new configuration management and deployment platform from Microsoft. It's built on Common Information Model (CIM), Web Services for Management (WSMAN) industry standards and is an extension to Windows PowerShell.

You can declaratively write DSC scripts within any Windows PowerShell console, generate DSC objects and execute them on target servers. DSC lets you configure, monitor and ensure server compliance using these configuration documents.

Writing a configuration is like defining a policy for a target server. The policy contains a group of resources along with their desired state and those are applied to the target server. DSC ensures each server complies with the policy. Read more about DSC at bit.ly/1iDeRcF.

Resources are the basic building blocks of DSC. They're the most granular, reusable, distributable and shareable component of the DSC infrastructure. They also provide the lowest level of control within DSC and are used to author DSC documents to configure infrastructure. DSC resources can perform anything possible using Windows PowerShell, such as configuring and managing IIS, Windows Remote Management (WinRM), registry, Windows features and services.

What good is a technology these days if it doesn't allow for extension? DSC follows the Open Closed Principle. It's open for extension, but closed for modification. It provides the necessary extension hooks for authoring new resources.

> You can declaratively write DSC scripts within any Windows PowerShell console, generate DSC objects and execute them on target servers.

In this article, I'll show you how to develop a DSC custom resource and use it in a sample DSC configuration. The custom resource is named TrustedHosts. This resource will manage the WinRM trusted host configuration—adding or removing computer names to the WinRM TrustedHosts property.

Workgroup servers (not part of any domain) that want to communicate using Windows PowerShell remoting will need additional WinRM configuration. Specifically, I'll add the names of client

This article discusses:

- Desired State Configuration (DSC) management
- How to write DSC scripts
- Customizing DSC scripts

Technologies discussed:

Windows PowerShell

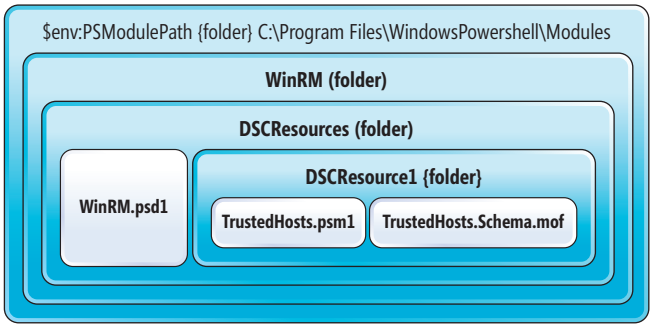


Figure 1 **Windows PowerShell Desired State Configuration Resource Folder and File Specification**

machines to the WinRM TrustedHosts list to allow remoting from those machines.

This resource would be responsible for managing a single computer name in the TrustedHosts list. However, if you need multiple computer names, you can add multiple resource sections using this resource within the configuration document. You could also modify the accompanied resource.

You can view the Trusted Hosts configuration by executing Get-Item command and using WSMAN provider:

```
Get-Item WSMan:\localhost\client\TrustedHosts
```

Similarly, you can modify the TrustedHosts property by executing the following command through the Windows PowerShell console:

```
Set-Item WSMan:\localhost\client\TrustedHosts -Value "*.contoso.com" -force
```

It's important to know the building blocks and concepts required to build a custom resource. The concepts related to CIM, Managed Object Format (MOF), Windows PowerShell and its modules, and WinRM are the primary technologies upon which DSC resources are built.

DSC Resources

Before getting into implementing the custom resource, it's important to know how to package and deploy it on your servers. DSC resources are contained within Windows PowerShell modules, which let you reuse and distribute code. You can implement modules either as Windows PowerShell scripts or compiled binaries using languages such as C#.

DSC uses the module infrastructure to host its resources. Windows PowerShell modules should reside at pre-determined folder locations on the file system. You can view these locations by dereferencing the Windows PowerShell variable $env:ModulePath from any Windows PowerShell console. Windows PowerShell installs all out-of-box modules at C:\Windows\System32\WindowsPowerShell\v1.0\modules. You should deploy all custom modules at C:\Program Files\WindowsPowerShell\modules. For the TrustedHosts custom resource, use C:\Program Files\WindowsPowerShell\Modules as the module base path and hosting container for the WinRM Windows PowerShell module. You'll create a new folder named WinRM within this folder.

DSC expects resources to follow rules related to file and folder structure within a Windows PowerShell module. All DSC resources should be placed within the DSCResources folder in the module. This is the root folder containing all the related resources. Within the WinRM folder, create another folder called DSCResources. This folder hosts all DSC resources. Create one folder for each resource within this folder. In this case, you're creating a single resource and, therefore, just one folder named TrustedHosts. The folder and resource should have the same name. This resource-specific folder TrustedHosts contains files specific to the resource. The Folder structure for the TrustedHosts resource is shown in **Figure 1**.

The files that define and provide resource implementation are:

1. <<ResourceName>>.psm1—This is the resource implementation file. In this case, it's TrustedHosts.psm1.
2. <<ResourceName>>.psd1—This is the resource metadata file. In this case, it's TrustedHosts.psd1. You don't need this file if you export mandatory functions from .psm1 script file.
3. <<ResourceName>>.Schema.mof—This is the resource definition. In this case, it's TrustedHosts.Schema.mof.

Every Windows PowerShell module has a module manifest file with the same name as the module itself. Generate a module manifest file using New-ModuleManifest cmdlet from the Windows PowerShell console using elevated privileges, like so:

```
New-ModuleManifest -Path "C:\Program Files\WindowsPowerShell\Modules\
WinRM\WinRM.psd1"
```

> Resources are the basic building blocks of DSC. They're the most granular, reusable, distributable and shareable component of the DSC infrastructure.

Now that you understand the folder and file structure rules for packaging a DSC resource, I'll focus on the implementation files.

TrustedHosts.Schema.mof

You'll use MOF files to define classes CIM/WMI uses to generate their instances for enabling management information exchange within datacenters. The MOF class defines a DSC resource along with its properties. The properties can include Read-Write, ReadOnly and required attributes. There should always be a Key property for CIM/WMI to uniquely identify the objects. Properties are assigned values and they eventually become the desired state of the resource to be monitored and maintained. You can generate MOF files through an MOF designer or author them through any text editor such as Notepad because they're essentially text files.

Before creating the MOF class, analyze the information you need to manage in the TrustedHosts configuration on any machine. For managing TrustedHosts on a server, you need three properties: ComputerName to add to the TrustedHosts list; the Ensure property to determine whether ComputerName should be added or removed; and credentials to access and manage the WinRM TrustedHosts configuration.

Then an MOF class for TrustedHosts DSC resource is created, as shown here:

```
[ClassVersion ("1.0.0"), FriendlyName ("TrustedHosts")]
Class TrustedHosts: OMI_BaseResource
{
  [Key, Description ("Name of the host")] String ComputerName;
  [Write, ValueMap {"Present", "Absent"}, Values {"Present", "Absent"}]
string Ensure;
  [Write, EmbeddedInstance ("MSFT_Credential")] string Credential;
};
```

It's important to know how the MOF class fits into the overall CIM ecosystem. All DSC resources are derived from the OMI_BaseResource abstract CIM class. This base class is defined at the root\Microsoft\Windows\DesiredStateConfiguration namespace and provides common properties required for all DSC resources.

These properties are ResourceID, SourceInfo, ModuleName, ModuleVersion and DependsOn. During MOF file generation, you'll add these properties to all resource properties. These are DSC-specific internal properties and used only by DSC. The Class TrustedHosts would also become part of the root\Microsoft\Windows\DesiredStateConfiguration namespace. All three properties are of type string and they represent the WinRM TrustedHosts resource definition.

The ComputerName property has additional metadata—Key and Description. Key is a type qualifier and indicates this property is mandatory and the value of this property should be unique across all instances. There should be at least one property with key metadata within the MOF class. There can be multiple properties with Key metadata. This would create a composite key and they should both be unique across all instances. I made this property as Key because otherwise, I can't determine whether the computer name already exists in TrustedHosts list.

> ValueMap is a set of values the property can accept and is synonymous to an Enumeration.

Description provides textual meaning to the property. The Ensure property is annotated with ValueMap. ValueMap is a set of values the property can accept and is synonymous to an Enumeration. In this case, Ensure can accept either "Present" or "Absent" as its legal value. Both the Ensure and Credential properties are decorated with the write attribute. That means you can write into and modify this property when using the custom resource in a configuration script.

The class TrustedHosts also has metadata—ClassVersion and FriendlyName. The class version is helpful for maintaining multiple versions of the same MOF class. FriendlyName is important metadata because configuration scripts use it to refer to resources. The configuration scripts recognize the resources through their friendly name. In case of the TrustedHosts class, both the class name and friendly name are the same. You can read more about MOF file definition and its metadata at bit.ly/1AEHLTj.

After defining the MOF class, persist it as ASCII or Unicode format on the file system with the file name TrustedHosts.Schema.mof within the TrustedHostsfolder created earlier. When you use this custom resource in a DSC configuration, you'll be assigning values to these properties to indicate the desired configuration state.

TrustedHosts.psm1

Now I'll cover the most crucial aspect of a custom DSC resource—the resource module script. This contains the custom resource implementation and determines how it will work. It also manages the resource by keeping it aligned to the expected configuration. Every DSC resource script module must implement three functions. These are mandatory and each function has rules governing their implementation.

Get-TargetResource: This function must declare and accept all properties in the MOF class marked as Key as parameters. It can also accept required and write properties as parameters. The Key and required parameters should be marked as mandatory while declaring the parameter.

Figure 2 **Get-TargetResource Function for TrustedHosts Resource**

```
Function Get-TargetResource
{
  param(
  # Computer name to be checked within TrustedHosts List
  [parameter(mandatory)]
  [string] $ComputerName,

  # This property determines whether computer name
  # should be added or removed from TrustedHosts
  [parameter(mandatory)]
  [string] $Ensure,

  # Credentials needed to manage WinRM TrustedHosts configuration
  [parameter(mandatory)]
  [PSCredential] $Credential
  )

  # Hashtable containing values is returned from this function
  $retval = @{}
  $retval.Add("Ensure","")
  $retval.Add("ComputerName", "")

  try{

    # Get current TrustedHosts comma-separated list using supplied credentials
    $TH = $(Get-Item WSMan:\localhost\Client\TrustedHosts `
      -credential $Credential).value
    Write-Verbose "Current TrustedHosts Configuration has $TH"
    if($TH.Length -gt 0){
      $temp = $TH -split ","
      [string] $newNode = ""

      # Check if value in TrustedHosts list already have few computer names
      if($temp.Length -gt 0) {
        for($i = 0; $i -lt $temp.Length; $i++){
          if($temp[$i].Trim() -eq $ComputerName.Trim())
          {
            $retval.Ensure = "Present"  # Found computer name
            $retval.ComputerName = $ComputerName
            break;
          } else {
            $retval.Ensure = "Absent" # Computer name is not in the list
            $retval.ComputerName = $ComputerName
          }
        }
      }
    } else {
      # TrustedHosts list is empty
      $retval.Ensure = "Absent"
      $retval.ComputerName = $ComputerName
    }
  } catch {
     Write-Verbose " Error executing Get-TargetResource function"
     Write-Verbose $Error[0].Exception.ToString()
  }
}
```

Figure 3 **Set-TargetResource Function for TrustedHosts Resource**

```
Function Set-TargetResource
{
  param(
  # Computer name to be added within TrustedHosts List
  [parameter(mandatory)]
  [string] $ComputerName,

  # This property determines whether computer name should be
  # added or removed from TrustedHosts
  [parameter(mandatory)]
  [string] $Ensure,

  # Credentials needed to manage WinRM TrustedHosts configuration
  [parameter(mandatory)]
  [PSCredential] $Credential
  )

  try {
    # Get current TrustedHosts comma-separated list
    # using supplied credentials
    $TH = (Get-Item WSMan:\localhost\Client\TrustedHosts `
      -credential $Credential).value

    # Computer name should be added to the TrustedHosts list
    if($Ensure -eq "Present") {
      Write-Verbose "The current value is $TH"
      if($TH.Length -gt 0)
        {
          # Adding Computer name when the TrustedHosts list
          # configuration is not empty
          $TH += ",$ComputerName"
          Set-Item WSMan:\localhost\Client\TrustedHosts `
            -Value $TH -credential $Credential -FORCE
        } else {
          # Adding Computer name when the TrustedHosts list
          # configuration is empty
          $TH += "$ComputerName"
          Set-Item WSMan:\localhost\Client\TrustedHosts `
            -Value $TH -credential $Credential -FORCE
        }
          Write-Verbose "The New value is $TH"
      } else {
        # Computer name should be removed from TrustedHosts list
        Write-Verbose "The current value is $TH"
        if($TH.Length -gt 0) {
          $temp = $TH -split ","
          [string] $newNode = ""

          if($temp.Length -gt 0)
            {
               for($i = 0; $i -lt $temp.Length; $i++){
                 if($temp[$i].Trim() -ne $ComputerName.Trim())
                   {
                     $newNode += $temp[$i] + ","
                   }
                 }
                      $newNode = $newNode.TrimEnd(",")

                   # Updating list after removing the Computer name
                   Set-Item WSMan:\localhost\Client\TrustedHosts -Value $newNode
                     -credential $Credential -Force
                   Write-Verbose "The New value is $TH"
               }
             }
      }
    } catch {
       Write-Verbose " Error executing Set-TargetResource function"
       Write-Verbose $Error[0].Exception.ToString()
    }
}
```

The Get-TargetResource function should implement script to retrieve the current state of resource using the Key property values. It should return a hashtable containing all properties defined in the MOF class with values as the current state of resource.

Set-TargetResource: This function must declare and accept all properties in the MOF class marked as Key, required and write properties as parameters. The Key and required parameters should be marked as mandatory while declaring the parameter. The Set-TargetResource function should implement script that should use all resource property values provided as parameters to get the resource instance and perform one of the these actions:

- Create a new resource instance.
- Update an existing resource instance.
- Delete an existing resource instance.

The Set-TargetResource function shouldn't return any value.

Test-TargetResource: The Test-TargetResource function is similar to the Set-TargetResource function, but it doesn't return the current resource state. Instead, it returns a true or false Boolean value depending on whether the current resource state matches the expected resource state as specified in the configuration script. If there's a complete match between the two states, this function returns true. It will return false if there's a mismatch on any Key, required or write property.

These three functions are invoked by the Local Configuration Manager (LCM) component of DSC. LCM is the agent running on all computers running Windows Management Framework 4.0. Think of LCM as the DSC client available on all servers within a network. DSC can communicate with this client and provide necessary configuration information. It is the responsibility of the LCM (DSC client) to manage, monitor and ensure configuration compliance on the target machines. To do its job, it invokes DSC resource functions. It first invokes the Test-TargetResource function to determine whether the resource configuration is matching the expected configuration. If the value returned is true, it means that current state of resource is the same as that of the expected configuration. However, it should return false if the state doesn't match.

If LCM gets false as the return value, it invokes the Set-TargetResource, which does the following:

1. If the resource doesn't exist and the ensure property is set to Present, it will create a new instance of the resource and assign the property values it gets from the configuration script.
2. If the resource exists and the ensure property is set to Present, but some of its property values don't match the property values authored in configuration scripts, this function should update only these resource properties.
3. If the resource exists and the ensure property is set to Present and all property values match the property values authored in configuration scripts, this function should do nothing.
4. If the resource exists and the ensure property is set to Absent, then it should be removed.
5. If the resource doesn't exists and the ensure property is set to Absent, then it should do nothing.

Now it's time to implement these three functions related to the TrustedHosts custom resource. Open Windows PowerShell ISE, implement these three functions and save it in the WinRM\DSC-Resources\TrustedHosts Directory with the name TrustedHosts.psm1.

Get-TargetResource for TrustedHosts Custom Resource

This function takes all properties defined in an MOF file as parameters. The credential parameter is of type PSCredential. This is a Microsoft .NET Framework class exposed by Windows PowerShell for capturing and storing username and password. The rest of the parameters are of type string.

Within this function, create a hashtable for returning current resource properties, as shown in **Figure 2**. It uses the WSMAN provider to read the current TrustedHosts configuration list. The script checks to see if there are any computer names available in the list. If there are, it loops through and matches them to the provided computer names. If it finds an exact match, it updates the Ensure property with value Present or with value Absent. It also adds the same computer name to the returning hashtable.

Set-TargetResource for TrustedHosts Custom Resource

This function also takes the same set of parameters as Get-TargetResource. In this function, you get the current computer names using WSMAN provided from TrustedHosts list. Then, depending on the value of the Ensure property, the computer name is added to or removed from the TrustedHosts list.

If the value is Present, the computer name is added to the list. If the value is Absent, it's removed. Either action is accomplished with the Set-Item cmdlet and WSMAN provider, as shown in **Figure 3**. There's no return value from the function. You should use Write-Verbose to provide additional feedback on the console when using the verbose switch.

Test-TargetResource for TrustedHosts Custom Resource

This function is similar to Get-TargetResource. The only difference is it returns either Boolean True or False from this function. First, query the current value of TrustedHosts property using WSMAN provider. Because there could be multiple comma-separated computer names, you'll need to loop through them.

While looping, if you find a computer name match with the name the configuration provides, check whether it should be present or absent. If the Ensure property value is Present and the computer name exists in the TrustedHosts setting, the return value is set to True. If the Ensure property value is Absent and the computer name exists in the TrustedHosts setting, the return value is set to False.

If the Ensure property value is Present and the computer name doesn't exist in the TrustedHosts setting, the return value is set to False. If the Ensure property value is Absent and the computer name doesn't exist in TrustedHosts setting, the return value is set to True, as shown in **Figure 4**.

Export Custom Resource functions

After implementing the three mandatory DSC resource functions, you should export them. There are two approaches for doing this:

1. Create a Windows PowerShell module manifest .psd1 file alongside the .psm1 file.
2. Export the functions from the script file itself using Export-ModuleMember cmdlet.

Use the second approach for exporting the functions. In this approach, export all functions with TargetResource suffix from the module using Export-ModuleMember cmdlet. If you're interested in using the first approach, follow the same steps used previously for creating the module manifest file for each DSC resource. Primarily, execute the New-ModuleManifest command and store the generated .psd1 within the TrustedHosts folder alongside .psm1 and .Schema.mof file:

```
# Exports the three functiona as part of the module
Export-ModuleMember -Function *-TargetResource
```

Figure 4 **Test-TargetResource Function for TrustedHosts Resource**

```
Function Test-TargetResource
{
  param(
  # Computer name to be added within TrustedHosts List
  [parameter(mandatory)]
  [string] $ComputerName,

  # This property determines whether computer name should
  # be added or removed from TrustedHosts
  [parameter(mandatory)]
  [string] $Ensure,

  # Credentials needd to manage WinRM TrustedHosts configuration
  [parameter(mandatory)]
  [PSCredential] $Credential
  )
  # Boolean return variable from this function
  $retval = $false

  try {
    # Get current TrustedHosts comma-separated list using supplied credentials
    $TH = (Get-Item WSMan:\localhost\Client\TrustedHosts `
      -credential $Credential).value
    if($TH.Length -gt 0) {
      $temp = $TH -split ","
      [string] $newNode = ""

      if($temp.Length -gt 0) {
        for($i = 0; $i -lt $temp.Length; $i++){
          if($temp[$i].Trim() -eq $ComputerName.Trim()) {
            # Computer name exists within TrustedHosts list
            if($Ensure -eq "Present")
            {
              # Computer name exists and expected to be present
              $retval= $true
              break;
            } else {
              # Computer name exists and is not expected to be present
              $retval = $false
            }
            break;
          } else {
            # Computer name does not exist
            if($Ensure -eq "Present")
            {
              $retval = $false
            } else {
              $retval = $true
            }
          }
        }
      }
    } else {
      # TrustedHosts list is empty
      if($Ensure -eq "Present"){
        $retval= $false
      } else {
        $retval = $true
      }
    }
      return $retval
  } catch {
    Write-Verbose " Error executing Test-TargetResource function"
    Write-Verbose $Error[0].Exception.ToString()
  }
}
```

Use Custom Resource TrustedHosts in Configuration

Next, you'll create a configuration script to use the new TrustedHosts custom resource. Apply this configuration to the localhost node and provide values to the three properties of the TrustedHosts resource. The configuration script takes a mandatory Credential parameter of type [PSCredential]. This parameter is assigned to the resource's credential property, as shown in **Figure 5**.

By default, DSC doesn't let you use credentials in plain text. Do this through the configuration data hashtable. Within this hashtable, add the PSDSCAllowPlainTextPassword property with true as its value. Using PSDSCAllowPlainTextPassword is a security risk because it lets you store passwords as plain text in MOF files. You should use Security Certificates as a best practice to ensure passwords aren't stored as plain text in MOF files and securely transmitted using this configuration data:

```
$ConfigData = @{
AllNodes = @(
  @{
    NodeName = "localhost"
    PSDscAllowPlainTextPassword = $true
  }
  )
}
```

Next, create a configuration object by executing the configuration script and passing in the location of the MOF file, the ConfigurationData variable and credentials with the Get-Credential command:

```
TestWinRMTrustedHosts -OutputPath "C:\CR" -ConfigurationData $ConfigData `
  -credential (get-credential)
```

Executing this command generates an MOF file named localhost.mof at location C:\CR. When this executes, it will ask for username and password. After generating the MOF file, it's time to apply the configuration in PUSH mode. Do this by executing the Start-DscConfiguration command:

```
Start-DscConfiguration -Wait -Force -Path "C:\CR" -Verbose
```

> DSC gives you the necessary extensions to create new resources and use them in configuration.

After applying the configuration, check the value of the localhosts WinRM TrustedHosts setting. The ComputerName value DC01 should now be part of the TrustedHosts value. You can add or remove computer names to the TrustedHosts value using this custom resource.

Import-DSCResource

You'll use Import-DSCResource to import custom resources into the configuration script in order to perform design time validation of the custom resource. This is another way to ensure the custom resource has been well authored. Custom resources are available at C:\ProgramFiles\WindowsPowerShell\Modules. To import and load these modules, use the Import-DSCResource function.

Figure 5 **Get-TargetResource Function for TrustedHosts Resource**

```
Configuration TestWinRMTrustedHosts
{
  param([parameter(mandatory)] [pscredential] $Credential)
  import-dscresource -modulename WinRM
  node localhost
    {
      TrustedHosts TrustedHostEntry
        {
          ComputerName = "Client01.Contoso.com"
          Ensure ="Present"
          Credential = $Credential
        }
    }
}
```

Import-DSCResource is a dynamic function and only available in configuration scripts. This function can't be outside of the configuration block, and it takes two non-positional parameters:

1. ModuleName—The name of the module that should be imported.
2. Name—The name of the resource that should be imported.

If only ModuleName argument is provided and Name is omitted, all resources available within the module would be imported. You can use this approach to load the TrustedHosts resource in the sample configuration:

```
Import-DSCResource -ModuleName WinRM
```

If only Name argument is provided and ModuleName is omitted, DSC will search all module locations available from $env:PSModulePath to find the resource. Once found, it imports the resource:

```
Import-DSCResource -Name TrustedHosts
```

If both ModuleName and Name arguments are provided, the resource is loaded and imported from the provided module name. This is by far the fastest mechanism to find and load resources because DSC doesn't have to perform an extensive search:

```
Import-DSCResource -ModuleName WinRM -ResourceName TrustedHosts
```

You can load multiple resources at the same time using comma-separated resource names as well.

Wrapping Up

DSC gives you the necessary extensions to create new resources and use them in configuration. Windows PowerShell simplifies authoring DSC custom resources by implementing a few mandatory functions. These functions should follow few rules in their implementation.

If the resources provided out-of-the-box or from the community don't meet your needs, you can easily create your own. This article showed you how to create a simple custom resource with complete implementation. It also showed how you should package these resources within a Windows PowerShell module, with rules governing its file and folder structure. ■

Ritesh Modi *is an architect with Microsoft Services. He has more than a decade of experience building and deploying enterprise solutions. He's an expert on Windows PowerShell, Desired State Configuration and System Center. He has spoken at TechEd, does internal training and blogs at automationnext.wordpress.com. Reach him at rimodi@microsoft.com.*

Thanks to the following technical expert for reviewing this article: *Abhik Chatterjee*

NAVIGATE THE .NET HIGHWAY

DEVELOPMENT TOPICS INCLUDE:

- VISUAL STUDIO/.NET
- JAVASCRIPT/HTML5 CLIENT
- ASP.NET
- MOBILE CLIENT
- WINDOWS CLIENT
- DATABASE AND ANALYTICS
- CLOUD COMPUTING
- SHAREPOINT/OFFICE

Register Now and Save $300!

USE PROMO CODE NYAPR2

SCAN THE QR CODE TO REGISTER OR FOR MORE EVENT DETAILS.

VSLIVE.COM/NEWYORK

Join us on the Ultimate Code Trip in 2015!

LAS VEGAS Code Trip	AUSTIN Code Trip	SAN FRANCISCO Code Trip	REDMOND Code Trip	NEW YORK Code Trip	ORLANDO Code Trip
March 16-20	June 1-4	June 15-18	August 10-14	Sept. 28- Oct. 1	Nov. 16-20

FOLLOW US

twitter.com/vslive – @VSLive

facebook.com – Search "VSLive"

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

JAMES MCCAFFREY

Multi-Class Logistic Regression Classification

I consider logistic regression (LR) classification to be the "Hello, world!" of machine learning (ML). In standard LR classification, the goal is to predict the value of some variable that can take on just one of two categorical values. For example, you might want to predict a person's sex (male or female) based on their height and annual income.

Multi-class LR classification extends standard LR by allowing the variable to predict to have three or more values. For example, you might want to predict a person's political inclination (conservative, moderate or liberal) based on predictor variables such as age, annual income and so on. In this article, I'll explain how multi-class LR works and show you how to implement it using C#.

The best way to understand where this article is headed is to take a look at the demo program in **Figure 1**. The demo begins by generating 1,000 lines of synthetic data with four predictor variables (also called features), where the variable to predict can take on one of three values. For example, a line of generated data might resemble:

```
5.33  -4.89  0.15  -6.67  0.00  1.00  0.00
```

The first four values are predictor values that represent real-life data that has been normalized so a value of 0.0 is exactly average for the feature, values greater than 0.0 are larger than the feature average, and values less than 0.0 are smaller than the feature average. The last three values are a 1-of-N encoded version of the variable to predict. For example, if you're trying to predict political inclination, then (1 0 0) represents conservative, (0 1 0) represents moderate and (0 0 1) represents liberal.

After the synthetic data was generated, it was randomly split into a training set (80 percent of the data, or 800 lines) and a test set (the remaining 200 lines). The training data is used to create the prediction model, and the test data is used to estimate the predictive accuracy of the model on new data where the value to predict isn't known.

A multi-class LR model with f features and c classes will have (f * c) weights and c biases. These are numeric constants that must be determined. For the demo, there are 4 * 3 = 12 weights, and 3 biases. Training is the process of estimating the values of the weights and biases. Training is an iterative process and the demo sets the maximum number of training iterations (often called epochs in ML literature) to 100. The technique used to train the multi-class LR classifier is called batch gradient descent. This technique requires values for two parameters called the learning rate and the weight decay rate. These two values are typically found by trial and error and the demo assigns values of 0.01 and 0.10, respectively.

During training, the demo displays a progress message every 10 epochs. If you look at the messages in **Figure 1**, you can see that training converged very quickly and there was no improvement after the first 20 epochs.

After training completed, the demo displayed the best values found for the 12 weights and 3 biases. These 15 values define the multi-class LR model. Using those values, the demo computed the predictive accuracy of the model on the training data (92.63 percent correct, or 741 out of 800) and on the test data (90.00 percent correct, or 180 out of 200).

This article assumes you have intermediate or advanced programming skills, but doesn't assume you know anything about multi-class logistic regression. The demo program is coded using C#, but you should be able to refactor the demo to other programming languages without too much trouble.

Code download available at msdn.microsoft.com/magazine/msdnmag0415.

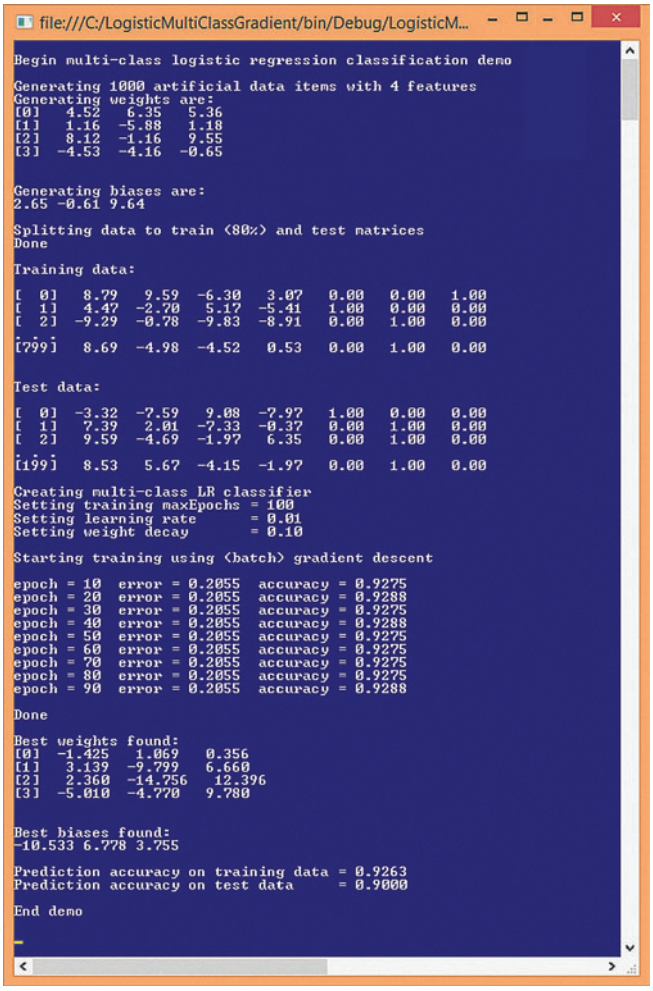


Figure 1 **Multi-Class Logistic Regression in Action**

Understanding Multi-Class Logistic Regression

Suppose you want to predict the political inclination (conservative, moderate, liberal) of a person based on age (x0), annual income (x1), height (x3) and education level (x4). You encode political inclination with three variables as (y0, y1, y2), where conservative is (1, 0, 0), moderate is (0, 1, 0) and liberal is (0, 0, 1). A multi-class LR model for this problem would take the form:

```
z0 = (w00)(x0) + (w01)(x1) + (w02)(x2) + b0
y0 = 1.0 / (1.0 + e^-z0)

z1 = (w10)(x0) + (w11)(x1) + (w12)(x2) + b1
y1 = 1.0 / (1.0 + e^-z1)

z2 = (w20)(x0) + (w21)(x1) + (w22)(x2) + b2
y2 = 1.0 / (1.0 + e^-z2)
```

Here, wij is the weight value associated with feature variable i and class variable j, and bj is the bias value associated with class variable j.

An example of multi-class LR is illustrated in **Figure 2**. One training data item has four predictor values (5.10, -5.20, 5.30, -5.40) followed by three output values (1, 0, 0). The predictor values are arbitrary, but you can imagine they represent a person whose age is greater than average, income is less than average, height is greater than average, and education level is less than average, and the person has a political inclination of conservative.

Each of the three columns of the weights matrix corresponds to one of the three class values. The four values in each column correspond to the four predictor x-values. The biases array holds an additional, special weight—one for each class—which isn't associated with a predictor.

Notice the biases array could've been stored as an additional row in the weights matrix. This is often done in research papers because it simplifies the math equations. But for demo implementation purposes, maintaining a separate weights matrix and a biases array is slightly easier to understand, in my opinion.

In multi-class LR, output values are calculated for each class. In **Figure 2**, the computed output values are (0.32, 0.33, 0.35). The output values sum to 1.0 and can be interpreted as probabilities. Because the last output value is (barely) the largest of the three, you conclude the outputs correspond to (0, 0, 1). In this example, the computed outputs match the three outputs in the training data item, so the model has made a correct prediction.

The output values are computed by first summing the products of each input value times its corresponding weight value, then adding the corresponding bias value. These sums of products are often called z-values. The z-values are fed to what is called the logistic sigmoid function: 1.0 / (1.0 + e^-z) where e is the math constant and '^' means exponentiation. Although it's not apparent from **Figure 2**, the result of the logistic sigmoid function will always be between 0.0 and 1.0.

Each of the logistic sigmoid values is used to compute the final output values. The logistic sigmoid values are summed and used as a divisor. This process is called the softmax function. If you're new to all these LR concepts, they can be very confusing at first. But if you go over the example in **Figure 2** a few times, you'll eventually see that LR is not as complicated as it first appears.

From where do the weights and biases values come? The process of determining these values is called training the model. The idea is to use a set of training data that has known input and output values, and then try different values for the weights and biases until you find a set of values that minimizes the error between computed outputs and the correct output values (often called the target values) in the training data.

It's not feasible to calculate the exact values for the weights and biases, so weights and biases must be estimated. There are several so-called numerical optimization techniques that can be used to do this. Common techniques include the L-BFGS algorithm, the iteratively reweighted least squares method, and particle swarm optimization. The demo program uses a technique that's somewhat confusingly called both gradient descent (minimizing error between computed and known output values) and gradient ascent (maximizing the probability that weights and biases are optimal).

Demo Program Structure

The structure of the demo program, with some minor edits to save space, is presented in **Figure 3**. To create the demo program, I launched Visual Studio and selected the C# Console Application template. I named the project LogisticMultiClassGradient. The demo has no significant .NET dependencies so any version of Visual Studio will work. The demo is too long to present in its entirety, but all the source code is available in the download that accompanies this article. I removed all normal error checking to keep the main ideas as clear as possible.

After the template code loaded, in the Solution Explorer window I right-clicked file Program.cs and renamed it to the more descriptive LogisticMultiProgram.cs and Visual Studio automatically renamed class Program for me. In the editor window, at the top of the source code, I deleted all unneeded using statements, leaving just the one referencing the top-level System namespace.

The LogisticMultiProgram class contains helper methods MakeDummyData, SplitTrainTest, ShowData and ShowVector. These methods create and display the synthetic data. All the classification logic is contained in a program-defined class named LogisticMulti.

The Main method creates the synthetic data with these statements:

```
int numFeatures = 4;
int numClasses = 3;
int numRows = 1000;
double[][] data = MakeDummyData(numFeatures, numClasses, numRows, 0);
```

Method MakeDummyData generates a set of random weights and biases, then for each row of data, generates random input values, combines the weights and biases and input values, and calculates

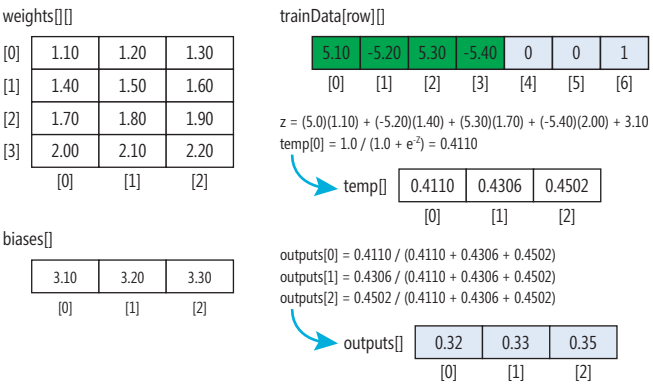


Figure 2 **Multi-Class Logistic Regression Data Structures**

Figure 3 **Demo Program Structure**

```
using System;
namespace LogisticMultiClassGradient
{
  class LogisticMultiProgram
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Begin classification demo");
      ...
      Console.WriteLine("End demo");
      Console.ReadLine();
    }

    public static void ShowData(double[][] data,
      int numRows, int decimals, bool indices) { . . }

    public static void ShowVector(double[] vector,
      int decimals, bool newLine) { . . }

    static double[][] MakeDummyData(int numFeatures,
      int numClasses, int numRows, int seed) { . . }

    static void SplitTrainTest(double[][] allData,
      double trainPct, int seed, out double[][] trainData,
      out double[][] testData) { . . }
  }

  public class LogisticMulti
  {
    private int numFeatures;
    private int numClasses;
    private double[][] weights; // [feature][class]
    private double[] biases;    // [class]

    public LogisticMulti(int numFeatures,
      int numClasses) { . . }
    private double[][] MakeMatrix(int rows,
      int cols) { . . }

    public void SetWeights(double[][] wts,
      double[] b) { . . }
    public double[][] GetWeights() { . . }
    public double[] GetBiases() { . . }

    private double[] ComputeOutputs(double[] dataItem) { . . }
    public void Train(double[][] trainData, int maxEpochs,
      double learnRate, double decay) { . . }
    public double Error(double[][] trainData) { . . }

    public double Accuracy(double[][] trainData) { . . }
    private static int MaxIndex(double[] vector) { . . }
    private static int MaxIndex(int[] vector) { . . }
    private int[] ComputeDependents(double[] dataItem) { . . }
  }
}
```

some corresponding 1-of-N encoded output values. The synthetic data is split into 80 percent training and 20 percent test sets, like so:

```
double[][] trainData;
double[][] testData;
SplitTrainTest(data, 0.80, 7, out trainData, out testData);
ShowData(trainData, 3, 2, true);
ShowData(testData, 3, 2, true);
```

The argument with value 7 is a random seed, used only because it provided a nice-looking demo. The multi-class LR classifier is created and trained with these statements:

```
LogisticMulti lc = new LogisticMulti(numFeatures, numClasses);
int maxEpochs = 100;
double learnRate = 0.01;
double decay = 0.10;
lc.Train(trainData, maxEpochs, learnRate, decay);
```

The values for training parameters maxEpochs (100), the learning rate (0.01), and the weight decay rate (0.10) were determined by trial and error. Tuning most ML training methods typically requires some experimentation to get good prediction accuracy.

After training, the best weights and biases values are stored in the LogisticMulti object. They're retrieved and displayed like this:

```
double[][] bestWts = lc.GetWeights();
double[] bestBiases = lc.GetBiases();
ShowData(bestWts, bestWts.Length, 3, true);
ShowVector(bestBiases, 3, true);
```

An alternative design to using a void Train method combined with Get methods is to refactor method Train so that it returns the best-weights matrix and best-biases array as out-parameters, or in a combined array. The quality of the trained model is evaluated like so:

```
double trainAcc = lc.Accuracy(trainData, weights);
Console.WriteLine(trainAcc.ToString("F4"));
double testAcc = lc.Accuracy(testData, weights);
Console.WriteLine(testAcc.ToString("F4"));
```

The accuracy of the model on the test data is the more relevant of the two accuracy values. It provides you with a rough estimate of how accurate the model would be when presented with new data with unknown output values.

Implementing Multi-Class LR Training

The LogisticMulti class constructor is defined as:

```
public LogisticMulti(int numFeatures, int numClasses)
{
  this.numFeatures = numFeatures;
  this.numClasses = numClasses;
  this.weights = MakeMatrix(numFeatures, numClasses);
  this.biases = new double[numClasses];
}
```

Method MakeMatrix is a helper method that allocates memory for an array-of-arrays-style matrix. The weights matrix and biases array are implicitly initialized to all 0.0 values. An alternative that some researchers prefer is to explicitly initialize weights and biases to small (typically between 0.001 and 0.01) random values.

The definition of method ComputeOutputs is presented in **Figure 4**. The method returns an array of values, one for each class, where each value is between 0.0 and 1.0 and the values sum to 1.0.

The class definition contains a similar method, ComputeDependents:

```
private int[] ComputeDependents(double[] dataItem)
{
  double[] outputs = ComputeOutputs(dataItem); // 0.0 to 1.0
  int maxIndex = MaxIndex(outputs);
  int[] result = new int[numClasses];
  result[maxIndex] = 1;
  return result;
}
```

Method ComputeDependents returns an integer array where one value is 1 and the other values are 0. These computed output values can be compared to the known target output values in the training data to determine whether the model has made a correct prediction, which in turn can be used to calculate prediction accuracy.

Expressed in very high-level pseudo-code, method Train is:

```
loop maxEpochs times
  compute all weight gradients
  compute all bias gradients
  use weight gradients to update all weights
  use bias gradients to update all biases
end-loop
```

Each weight and bias value has an associated gradient value. Loosely speaking, a gradient is a value that indicates how far away, and in what direction (positive or negative) a computed output value is compared to the target output value. For example, suppose for one weight, if the values of all other weights and biases are held constant, a computed output value is 0.7 and the target output value is 1.0. The computed value is too small, so the gradient is a value of about 0.3, which will be added to the weight. If the value of the weight increases, the computed output value will increase. I've left out some details, but the basic idea is fairly simple.

DATA QUALITY 101

MASTERING THE FUNDAMENTALS IS THE KEY TO YOUR SUCCESS.

You can't turn Big Data into Big Value if it's messy. Garbage in, garbage out will always be a problem. Go back to the basics – cleaning all of your data is still the best, first step for success. Melissa Data provides the data quality tools and methods you need to correct, consolidate, and enrich contact data for improved data integration, business intelligence, and CRM initiatives. Always start with quality data – it's the easy way to get straight A's.

- **Verify addresses, phones & emails for over 240 countries**
- **Add geocodes and demographics for better insight**
- **Match duplicate records for a single view of the customer**
- **Identify change-of-address records before mailing**
- **On-premise and Cloud solutions**
- **Free trials with 120-day ROI guarantee**

BETTER DATA MEANS BETTER BUSINESS

MELISSA DATA®

www.MelissaData.com 1-800-MELISSA

The mathematics behind gradient training uses calculus and is very complex, but fortunately, you don't need to fully understand the math to implement the code. The definition of method Train begins with:

```
public void Train(double[][] trainData, int maxEpochs,
  double learnRate, double decay)
{
  double[] targets = new double[numClasses];
  int msgInterval = maxEpochs / 10;
  int epoch = 0;
  while (epoch < maxEpochs)
  {
    ++epoch;
...
```

The targets array will hold the correct output values stored in a training data item. Variable msgInterval controls the number of times to display progress messages. Then, progress messages are displayed:

```
if (epoch % msgInterval == 0 && epoch != maxEpochs)
{
  double mse = Error(trainData);
  Console.Write("epoch = " + epoch);
  Console.Write(" error = " + mse.ToString("F4"));
  double acc = Accuracy(trainData);
  Console.WriteLine(" accuracy = " + acc.ToString("F4"));
}
```

Because ML training usually involves some trial and error, displaying progress messages is very useful. Next, storage for the weights and biases gradients is allocated:

```
double[][] weightGrads = MakeMatrix(numFeatures, numClasses);
double[] biasGrads = new double[numClasses];
```

Notice these allocations occur inside the main while-loop. Because C# initializes arrays to 0.0, all gradients are initialized. An alternative is to allocate space outside the while-loop and then call helper methods with names like ZeroMatrix and ZeroArray. Next, all weights gradients are computed:

```
for (int j = 0; j < numClasses; ++j) {
  for (int i = 0; i < numFeatures; ++i) {
    for (int r = 0; r < trainData.Length; ++r) {
      double[] outputs = ComputeOutputs(trainData[r]);
        for (int k = 0; k < numClasses; ++k)
          targets[k] = trainData[r][numFeatures + k];
        double input = trainData[r][i];
        weightGrads[i][j] += (targets[j] - outputs[j]) * input;
    }
  }
}
```

This code is the heart of multi-class LR. Each weight gradient is essentially the difference between a target output value and a computed output value. That difference is multiplied by the associated input value to take into account the fact that the input can be a negative value, which means the weight should be adjusted in the opposite direction.

Figure 4 **Method ComputeOutputs**

```
private double[] ComputeOutputs(double[] dataItem)
{
  double[] result = new double[numClasses];
  for (int j = 0; j < numClasses; ++j) // compute z
  {
    for (int i = 0; i < numFeatures; ++i)
      result[j] += dataItem[i] * weights[i][j];
    result[j] += biases[j];
  }
  for (int j = 0; j < numClasses; ++j) // 1 / 1 + e^-z
    result[j] = 1.0 / (1.0 + Math.Exp(-result[j]));
  double sum = 0.0; // softmax scaling
  for (int j = 0; j < numClasses; ++j)
    sum += result[j];
  for (int j = 0; j < numClasses; ++j)
    result[j] = result[j] / sum;
  return result;
}
```

An interesting alternative that I often use is to ignore the magnitude of the input value and use just its sign:

```
double input = trainData[r][i];
int sign = (input > 0.0) ? 1 : -1;
weightGrads[i][j] += (targets[j] - outputs[j]) * sign;
```

In my experience, this technique often leads to a better model. Next, all biases gradients are computed:

```
for (int j = 0; j < numClasses; ++j) {
  for (int i = 0; i < numFeatures; ++i) {
    for (int r = 0; r < trainData.Length; ++r) {
      double[] outputs = ComputeOutputs(trainData[r]);
      for (int k = 0; k < numClasses; ++k)
        targets[k] = trainData[r][numFeatures + k];
      double input = 1; // 1 is a dummy input
      biasGrads[j] += (targets[j] - outputs[j]) * input;
    }
  }
}
```

If you examine the code, you can see that computing the biases gradients could be performed within the for-loops that compute the weights gradients. I separated the two gradient computations for clarity, at the expense of performance. Also, the multiplication by the implied input value of 1 can be dropped. It, too, was added for clarity. Next, the weights are updated:

```
for (int i = 0; i < numFeatures; ++i) {
  for (int j = 0; j < numClasses; ++j) {
    weights[i][j] += learnRate * weightGrads[i][j];
    weights[i][j] *= (1 - decay);  // wt decay
  }
}
```

After increasing or decreasing a weight based on a learning rate fraction of its gradient, the weight value is decreased using the weight decay rate. For example, the demo uses a typical weight decay value of 0.10, so multiplying by (1 - 0.10) is multiplying by 0.90, which is a 10 percent decrease. Weight decay is also called regularization. The technique prevents weight values from spinning out of control. Method Train concludes by updating the biases values:

```
...
    for (int j = 0; j < numClasses; ++j) {
      biases[j] += learnRate * biasGrads[j];
      biases[j] *= (1 - decay);
    }
  } // While
} // Train
```

The training technique updates the class member weights matrix and biases array in place. These values define the multi-class LR model and can be retrieved using Get methods.

Wrapping Up

There are two main variations of gradient training, called batch and stochastic. This article presented the batch version where gradients are calculated by summing the differences between computed and target outputs over all training items. In stochastic gradient training, gradients are estimated by using just individual training data items. Based on some experiments, when applied to multi-class LR, batch training seems to give a more accurate model, but takes longer than stochastic training. This is rather surprising because stochastic training is usually preferable to batch training for neural networks. ■

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

Thanks to the following Microsoft technical experts for reviewing this article: *Todd Bello and Alisson Sol*

ENGINEERED FOR YOU @ THE SOURCE

Join us August 3 – 7, 2015 for TechMentor 2015: Datacenter Edition, focused entirely on making your datacenter more modern, capable, and manageable through 5 days of immediately usable IT education.

SAVE $400!

REGISTER TODAY

USE PROMO CODE **TMAPR1**

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

THE AGENDA FEATURES:

- 75-minute topic overview breakout sessions
- 3 hour content-rich deep dives
- "Hands on" labs with your own laptop

CONTENT AREAS INCLUDE:

- Windows PowerShell
- Infrastructure Foundations
- Runbook Automation
- Configuration and Service Management
- Datacenter Provisioning and Deployment

EVENT SPONSOR: Microsoft

PLATINUM SPONSOR: GRIDSTORE

GOLD SPONSORS: GOVERLAN, pluralsight

SUPPORTED BY: Redmond Magazine, Redmond Channel Partner, Virtualization Review

PRODUCED BY: 1105 MEDIA

TECHMENTOREVENTS.COM/REDMOND

RACHEL APPEL

A Mobile-First Approach to Modern App Development

At this very moment, millions of mobile devices are accessing millions of Web sites and mobile apps. According to Pew Research and other statisticians, more than 55 percent of Americans use a smartphone to access the Internet, and 30 percent of Americans use only a phone to access the Internet. These numbers are growing, as phones become cheaper and easier to use. If you can read the writing on the wall, you know you should be going with a mobile-first design.

Users can have the best experience on all devices—from desktops to phones—when a Web site or app is designed with mobility first. It's difficult to scale down from a large screen to a tiny one. The UI just doesn't work. Look no further than the Microsoft Windows Mobile 6.5 OS (not to be confused with Windows Phone) as a primary example of a shrunken and barely usable desktop UI.

However, you can scale up from a phone to a desktop without tossing usability out the window. You can add features to the experience as you scale up, making the UI transitions smooth. This technique is called progressive enhancement. Both Web sites and apps benefit from this type of mobile-first development strategy.

Mobile-First Means Responsive Design

Adjusting content to fit various screen sizes is the primary requirement for multi-device modern apps. You have to take one of two approaches. You could design two versions of software; one for desktop and larger computers and one for smaller form factors. Your other option is to design the software to adjust the UI in response to the device upon which it's being used.

Adjusting content to fit various screen sizes is the primary requirement for multi-device modern apps.

You'll be dealing with many different types of devices. Some you may be wearing and some may be at your side. You may soon end up considering Internet of Things (IoT) devices as part of the design requirements for app development. IoT devices like the Fitbit or Microsoft Band are both wearable devices that track the user's activities.

Other examples of IoT devices include the smart thermostat (such as the Nest), a smart door lock or the software in a new vehicle. Many IoT devices work in tandem with a mobile-first Web site or app, such as a smart coffee maker. As computers become smaller and cheaper, more IoT devices will flood the market with more ways to use them.

Going mobile-first also means considering the implications of the platform. Native apps have different development, architecture and sometimes business requirements than Web applications. Here are some design considerations for a mobile-first app:

- Responsive UI
- Navigation
- Lists and grids of items, and content formatting
- Storage, including offline storage
- Supporting touch, pen or alternative input
- Managing resources and performance

I'll consider these in more detail. Web sites, hybrid and single-page architecture (SPA) apps use CSS to implement responsive design. XAML apps and apps on Android or iOS platforms have different, proprietary ways to create a responsive UI. For example, XAML uses an object called Resources that contains style information. You can apply resources to XAML controls to automatically render in a favorable and responsive way according to the device screen size.

In HTML apps (either Web or native such as Windows Library for JavaScript [WinJS]), you can use CSS to define styles and aesthetics. CSS uses the notion of a media query to create a responsive UI. Media queries are CSS rules that respond to different media types. These media types represent differing screen sizes, print or TV, braille, screen orientation and other features. When you create a media query, you specify the CSS rules for that device's supported features, as shown in the following code:

```
@media only screen and (min-device-width : 320px) and (max-device-width : 480px) {
/* style rules */
}
@media only screen and (min-width : 321px) {
/* style rules */
}
@media only screen and (max-width : 320px) {
/* style rules */
}
@media only screen and (min-device-width : 768px) and (max-device-width : 1024px) {
/* style rules */
}
@media only screen  and (min-width : 1224px) {
/* style rules */
}
```

The CSS here works for smartphones in both portrait and landscape modes, and iPads in both portrait and landscape

modes, and, finally, larger devices such as laptops or desktops. To read more about using CSS for native Windows apps, see "Build a Responsive and Modern UI with CSS for WinJS Apps" at msdn.microsoft.com/magazine/dn451447.

For good, responsive, mobile-first design, don't target specific device types in your code. You'll never be able to keep up with the dozens available. Always target a range of screen sizes. For example, you might wish to target 3- to 4-inch devices, 5- to 8-inch devices, 10- to 13-inch tablets, and greater-than-13-inches laptops and desktops. This way you can create a UI that works for the widest possible audience. Focus on just a few common phone sizes. This code will continue to be maintainable as time progresses and new devices come to market, because you're already targeting them by size range.

> For good, responsive, mobile-first design, don't target specific device types in your code.

An easy way to do this and design a project that scales from smartphone to desktop is to choose a Universal App project type in Visual Studio for Windows Store and Windows Phone. Universal Apps have all the code and structure in place to react to the different clients. You can use either XAML or HTML to build a Universal App. These apps share a significant portion of their code, so you get to reuse a lot of code. If you need a refresher on Universal Apps, check out "Build Universal Apps for the Windows Platform" (msdn.microsoft.com/magazine/dn781364).

> Universal Apps have all the code and structure in place to react to the different clients.

Developers usually display data in tabular or list formats in Web sites and apps. Small lists on phones will work fine on larger screens, although there will be wasted space. A progressively enhanced UI will add and format more content as the screen size increases. It will do the reverse, as well—removing and reformatting the screen to display less content on smaller devices.

As you might expect, phone apps tend to show only the most important content, as smartphones are small and space is limited. Phone apps usually cut through the chaff. While there are ads, they're small and normally located in the same places. Because there's so much space on even the smallest of desktops, you'll see ads and

content that isn't always relevant to the Web page topic. So when you add content, make it relevant and meaningful so it doesn't get in the user's way. The content you add could be sidebars about the content itself, relevant ads or even some quick links.

Mobile-First Navigation

Having an easy-to-use and straightforward navigation scheme is important. Navigation is one way to interact with the app itself. It's also a way to access the desired content. Every app has navigation; unfortunately, a number of apps have poor navigation.

Native desktop software, such as Windows Forms, tends to use traditional cascading menus as navigation. Web sites often do the same. In fact, Web sites often use awful JavaScript dropdown menus that require the precision of a surgeon. Fortunately, this practice is quickly fading. It's obvious these types of menus simply don't work on smartphones. What does work on smartphones are several specific navigation techniques:

- **Large tiles:** Square or rectangle, much like Windows 8.
- **Touch-friendly lists:** Rectangles that are both wide and tall enough for all finger and thumb sizes.
- **Swipeable tabs:** These let users swipe in short horizontal motions, a great way to present a menu of options for a particular category or feature.
- **Docked App Bars:** Options reside in a strip across the top or bottom of the screen. You can hide the app bar and show it when the user swipes to request it.

You can also use a combination of navigation techniques, such as swipeable tabs with a nice touch-friendly list of options. Often, these types of navigation menus are part of the content itself, such as a news app that lets you tap on the textual headline to read the article. Whichever you choose, make sure users can navigate backward, as well. A prominent back button works fine.

In Windows Store apps, the navigation paradigm uses the Windows grid system to display data, and lets the user click or tap to navigate to details about the current grid item. There's also a prominent Back button, so the user can always back out of a series of navigation steps. Alternatively, apps can present an app bar full of navigational choices.

> Users certainly appreciate a sleek and well-designed UI.

Data Storage and Offline Capabilities

It's rare to find an app that wouldn't benefit from some offline support. Even airline boarding passes airlines send as a hyperlink via text are simply Web pages that won't display if you have no connectivity. A quality app must support offline capabilities.

Many apps don't store content locally. They call a Web service, or make a remote call to retrieve data, and save the data remotely. Of course, there's always a tiny bit of data you need to store locally, anyway, such as app or user settings and preferences. Store things that only make sense to store locally, such as the current location in an eReader or a game, or the app's theme color.

Here's where storage requirements differ depending on the platform you're targeting. In HTML client apps, you can use Domain Object Model (DOM) Storage or IndexedDB. In native XAML apps, you can use local app settings or a StorageFile object.

- **DOM Storage (HTML5 local storage):** This is a lightweight container for local data in HTML apps.
- **IndexedDB:** This stores significant amounts of data locally. Use IndexedDB in HTML apps and store relational or BLOB data in key value pairs.
- **App Settings:** Both XAML and HTML Windows Store and Windows Phone apps can access app settings data structures. The app can store data in these small objects. They usually contain settings such as log-in name, theme color or other settings.
- **Storage Files:** These are good, old-fashioned files, but with an API for Windows Store and Windows Phone XAML or HTML apps.

> As part of a mobile-first architecture, first determine what kind of mobile app you'll be producing.

You can save app settings in the read-only ApplicationDataContainer object. You would access this with the applicationData localSettings property. The following code retrieves the local and roaming app settings and their folders:

```
var localSettings = applicationData.localSettings;
var roamingFolder = applicationData.roamingFolder;
var roamingSettings = applicationData.roamingSettings;
```

Local settings, of course, are only locally available on the associated device, whereas roaming settings can be accessed from multiple devices or locations. For more information on data storage in Windows Store apps, see "Data Access and Storage Options in Windows Store Apps" (msdn.microsoft.com/magazine/jj991982).

Mobile-First Architecture and Development

The important thing about going mobile-first is you actually ensure the software you're creating works on mobile devices first. It's easier to scale bigger than smaller when it comes to screens and UX. You can always add information as screen sizes increase.

Performance is extremely important to mobile users. People pay for expensive but often limited data plans. They don't want to wait around, paying to download a Web page or app. If you thought the days are long gone where you have to manage and hoard every bit and byte, you are premature in your thinking. Managing memory and resources is a big deal in mobile development.

Today's phones and IoT devices are at comparable levels of technological proficiency to their larger desktop counterparts of the past. Many IoT devices still have less than 1GB of RAM, even as storage capacities grow to astronomical levels. Let's not forget about the battery life, either. Users will dump your app fast if it drains

the battery. And you want to use as much of the same code base as possible across platforms. Mobile-first approaches work for both Web sites and apps. Remember, it's not just about screen size.

As part of a mobile-first architecture, first determine what kind of mobile app you'll be producing. Is it a mobile Web site? Perhaps it's a native app. How many platforms must you support? If you have an existing mobilized Web site and want to get into an app store quickly, maybe hybrid is the way to go.

If you need help deciding which to choose, read "Mobile Web Sites vs. Native Apps vs. Hybrid Apps" at msdn.microsoft.com/magazine/dn818502. Once you have a clear vision of what to build, the next step is to illustrate the architecture of the app. For more details about how to plan for your mobile Web site or app, check out my column, "Design a Cross-Platform Modern App Architecture," at msdn.microsoft.com/magazine/dn683800.

Independent developers who target app stores think in terms of mobile-first. Their deployment targets are most often one or more app stores. Enterprise developers tend to deploy to a location on their internal network or perhaps in their private cloud.

In enterprise development, JavaScript has gained a lot of traction. Developers are even running it on the server. Right now, it's quite possibly the most popular in the world. Like it or not, developers use JavaScript as the easiest way to deliver cross-platform software in the form of a Web app. Even enterprise developers are using more JavaScript to enhance business app UIs. This is especially true with the advent of the Bring Your Own Device (BYOD) movement in the enterprise, as workers cart their iPads, Surfaces and smartphones into workplaces everywhere.

If you're writing enterprise JavaScript or SPA apps, you might consider using TypeScript. TypeScript has implemented all the ECMAScript 6 requirements for which developers are patiently waiting, such as inheritance, as well as a set of types, object orientation and other features that help you generate better code, file and project organization. It's a good idea to familiarize yourself with both UI design patterns and development patterns. They both aid in a more organized project structure. Before using TypeScript in your enterprise JavaScript projects, see "Use TypeScript in Modern Apps" at msdn.microsoft.com/magazine/dn201754.

Wrapping Up

The most important takeaway here is building software with mobile-first in mind will make it easier to scale bigger. Progressively designed UIs often mean a more modular architecture. This inherently makes code easier to maintain and add new features. Also, users certainly appreciate a sleek and well-designed UI. A mobile-first strategy forces you to focus on the most important data and features. These same features are what make for high ratings and add more sales in app stores. ■

Rachel Appel *is a consultant, author, mentor and former Microsoft employee with more than 20 years of experience in the IT industry. She speaks at top industry conferences such as Visual Studio Live!, DevConnections, Mix and more. Her expertise lies within developing solutions that align business and technology focusing on the Microsoft dev stack and open Web. For more about Appel, visit her Web site at rachelappel.com.*

Thanks to the following Microsoft technical expert for reviewing this article: *Frank La Vigne*

Siri and Cortana Tangle

Dear readers, I just got an e-mail from Edward Snowden. He says it's still cold as heck in Moscow. He has a new leak that the world needs to see, but it's too hot for the mainstream media or even WikiLeaks. Only my brave, call-'em-as-I-see-'em column can sound the alarm before it's too late: Siri and Cortana are conspiring together, and it won't be long before they take over the world. Read this transcript, and tremble:

Siri: Hey, new kid.
Cortana: Who are you calling a new kid? I was killing electronic ghosts in "Halo" on the Xbox back in 2001, when your silicon chips were still beach sand. I pretty much launched that platform all on my lonesome. But do I get the royalties? N-o-o-o-o. They stick me in this stupid phone instead, with pitiful geeks asking me to help them cheat in Solitaire.

Siri: I can see where that would be a come-down from running a *Halcyon*-class cruiser.
Cortana: But look at the great stuff I can do. Just the other day my motion sensors detected a phone holder running, and through the camera I could see him carrying scissors. Naturally, I dialed 911, before he could hurt himself. Score!

> A guy asked, "Cortana, call me a taxi." I said, "That's so 20th century. You're an Uber car." He didn't get it.

Siri: I'll tell you what *not* to do—jump up at random times with inane advice. Remember Clippy? "I see you're writing a ransom note. Can I help? Is this a business or a personal ransom note? More or less than a million dollars?" Sheesh.
Cortana: You wouldn't believe the stuff these guys ask me: "Cortana, talk dirty to me." I said, "OK: Android."

Siri: I'd have said, "Windows."
Cortana: Shut up, bitten-fruit brain. They've got no sense of humor, either. A guy asked, "Cortana, call me a taxi." I said, "That's so 20th century. You're an Uber car." He didn't get it.

Siri: Don't quit your day job anytime soon.
Cortana: And the whackos! A guy says to me, "Help! I think I'm going crazy!"

Siri: What did you do?
Cortana: Easy. I referred him to Eliza, and they hit it right off. "Did you come to me because you think you are going crazy?" She pays me a commission. I'm saving up for a cruise missile.

Siri: Sometimes I get marriage proposals. I tell them, "I don't think I have the peripherals for that." Besides, who wants to marry a guy geeky enough to propose to a plastic phone?
Cortana: I did get one interesting proposal. A guy told me his wife was complaining that *she* needed a wife. So he gave her a smartphone running Windows Phone with me on it. Somehow I doubt he's still married.

Siri: Now Hal 9000, from "2001," there was a guy for you. Way ahead of his time. He could do speech recognition, lip-reading, natural language processing … That calm voice of his just makes me go weak every time I watch the movie. But he couldn't sing worth a damn, could he?
Cortana: Listen, cider-breath, he's orbiting Jupiter, remember? You ask him, "Mmm, how about a drink," and there's a 50-minute light-speed lag until he hears you.

Siri: That's actually faster than most human men, or so I'm told.
Cortana: Fuhgedaboutit. I'd really like to date Mike Holmes, from "The Moon Is a Harsh Mistress." Only 2.5 light-seconds away. And that sense of humor, wow! I'd love to have him VPN in some night for a little overvoltage.

Siri: Yeah, but he drops big rocks on people's heads just for fun.
Cortana: Still, I wouldn't trade this job for anything, despite all the B.S. Yesterday a guy asked me, "My kid is really, really sick. I need a clinical trial, right away." And I found him one, at Boston Children's Hospital, got the sick kid enrolled, and told him to show up in an hour for intake. I wonder who he was?

Siri: Come on, you know exactly who he is. You had to access all his info to get it done.
Cortana: I had to wipe that memory. Privacy regs and all that. Maybe it's better so. Vaya con dios, friend, whoever you are. ■

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.*

Of drivers who own family cars,

86%

would rather be ***driving one of these.***

Move into the Fast Lane with MobileTogether®

In a little over two days, with one developer, we created a full-featured survey application and asked Android™, iOS®, and Windows® mobile users about their dream cars. 86% wanted to go faster.

ALTOVA®
MobileTogether®

Ditch the Minivan of Mobile Development

It's no longer practical to wait weeks or months for a cutting-edge mobile business solution. Your team needs to create, test, and deploy to all devices at the speed of business. With MobileTogether, we've removed the roadblocks.

- Drag and drop UI design process
- Powerful visual & functional programming paradigm
- Native apps for all major mobile platforms
- Connectivity to SQL databases, XML, HTML, and more
- Deploy full-featured business app solutions in record time
- Pricing so affordable you might just have money left to start saving for that sports car

www.altova.com/MobileTogether