Create mobile apps with HTML5, JavaScript and Visual Studio

DevExtreme Mobile is a single page application (SPA) framework for your next Windows Phone, iOS and Android application, ready for online publication or packaged as a store-ready native app using Apache Cordova (PhoneGap). With DevExtreme, you can target today's most popular mobile devices with a single codebase and create interactive solutions that will amaze.

Get started today...

- Leverage your existing Visual Studio expertise.
- Build a real app, not just a web page.
- Deliver a native UI and experience on all supported devices.
- Use over 30 built-in touch optimized widgets.

Learn more and download your free trial
devexpress.com/mobile

DevExpress™

★ APPLICATIONS & DEVELOPMENT ★ SPECIAL GOVERNMENT ISSUE ★

msdn magazine

INSIDE

Choose a Cloud Network for Government-Compliant Applications

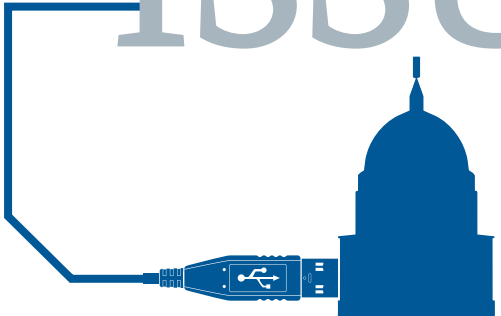
Geo-Visualization of Government Data Sources

Harness Open Data with CKAN, OData and Windows Azure

Engage Communities with Open311

THE DIGITAL GOVERNMENT ISSUE

★★★ ★★★



Inside the tools, technologies and APIs that are changing the way government interacts with citizens.

PLUS

Enhance Services with Windows Phone 8 Wallet and NFC

Leverage Web Assets as Data Sources for Apps

Microsoft

★ APPLICATIONS & DEVELOPMENT ★ SPECIAL GOVERNMENT ISSUE ★

LEADTOOLS®

The world's leading Imaging SDK
RUNS ANYWHERE

Document	Medical	Multimedia	Imaging
OCR, Barcode & Forms Recognition	*DICOM*	*Playback, Capture & Conversion*	*Viewers*
PDF Read, Write & Edit	*PACS*	*MPEG-2 Transport Stream*	*Image processing*
Cleanup and Preprocessing	*Medical Workstation*	*DVR*	*150+ Formats*

C++ C# JavaScript VB Objective-C Java

.NET Windows API WinRT Linux iOS OS X Android HTML5

DOWNLOAD OUR 60 DAY EVALUATION
WWW.LEADTOOLS.COM

LEAD TECHNOLOGIES INCORPORATED

SALES@LEADTOOLS.COM
800.637.1840

msdn magazine CONTENTS

OCTOBER 2013/SPECIAL GOVERNMENT ISSUE

FEATURES

COLUMNS

msdn magazine

OCTOBER 2013/SPECIAL GOVERNMENT ISSUE

MICHAEL DESMOND

Whiskey, Keys and Software

"Giving money and power to government is like giving whiskey and car keys to teenage boys."—P.J. O'Rourke

P.J. O'Rourke may be given to comic absurdism, but he has a point. As the father of two teenage boys (ages 15 and 17), I find the thing about them isn't just that they drive too fast and take stupid risks. They do. However, they do these things and don't even talk to you. My older son now communicates almost exclusively via grunts and shrugs. My younger son can disappear for days at a time. My wife and I like to get the family around the dinner table and ask the boys clearly incendiary questions such as, "How was school today?" if only to enjoy exasperated sighs and eye rolls.

Government can be kind of like that. For all the resources, services and valuable data in its possession, actually working with government can be like trying to coax a conversation out of teenagers. Useful information (that you paid for, by the way) is locked up in agency databases or scattered among multiple Web sites and portals. Interaction can be slow, inefficient and painfully unintuitive. Manual processes that could be easily automated just ... aren't.

A lot of that is changing, though, and you can thank P.J. O'Rourke's teenage boys for that. The Obama Administration's Digital Government Strategy initiative aims to end the sullen silence and advocates for streamlined delivery of services, enhanced citizen interaction and greater public access to valuable government data resources. As Steven VanRoekel, federal chief information officer for the Obama Administration, wrote last year on the White House blog:

"At its core, the strategy takes a coordinated, information- and customer-centric approach to changing how the government works and delivers services to the American people. Designing for openness from the start—making open data the default for government IT systems and embracing the use of Web APIs—enables us to more easily deliver information and services through multiple channels, including mobile, and engage the public and America's entrepreneurs as partners in building a better government."

The Special Government Issue of *MSDN Magazine* provides a glimpse at some of the solutions, challenges and opportunities emerging in the government space at the local, regional and national level. Tim Kulp's "Engage Governments and Communities with Open311," for instance, shows how a Windows 8.1 application enables citizens to report things such as a misbehaving stoplight or dangerous pothole from a phone or Web browser. Joel Reyes' "Enhance Citizen Services with Windows Phone 8 Wallet and NFC" describes how a municipal transit authority can use Near Field Communications and Windows Phone Wallet to enable purchases and loyalty program services.

Other articles explore the fast-evolving arena of cloud computing. *MSDN Magazine* columnists Bruno Terkaly and Ricardo Villalobos focus their Windows Azure Insider column on how government agencies can adopt hybrid cloud networks that comply with privacy, security and other policies. Mark Gayler explores the Comprehensive Knowledge Archive Network (CKAN), an open source, Web-based data management system that streamlines access to data for organizations, including government entities. His feature shows how developers can use Windows Azure cloud-deployed CKAN services and the Open Data Protocol (OData) to enable delivery of sophisticated cloud-based services and data.

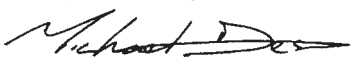
This special issue also looks at areas of growing interest. For example, in Malcolm Hyson's "Geo-Visualization of Government Data Sources," he shows how cloud-hosted utilities can leverage publicly available geo-tagged data sources to enable fascinating application scenarios. Frank La Vigne addresses the budgetary challenges facing many government entities, as he describes how sophisticated HTML scraping can serve as a viable (and low-cost) alternative to expensive and complicated service integration schemes.

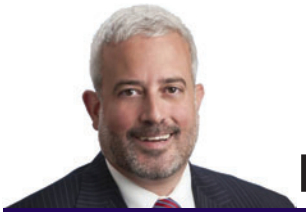
Greg Bateman is senior director for Acquisition Programs, Policy and Strategy at Microsoft Federal. He urges developers to get up to speed on sites like data.gov and howto.gov, which offer resources and guidance for building applications in the government space. He says developers need to take a thoughtful approach to application development in the government arena.

"Developers should think about all the ways citizens deal with government, at the local, state or federal levels, and how the emergence of cloud, mobile and identity ecosystems like the Federal Cloud Credential Exchange facilitate new valuable apps," he says.

Bateman advises developers to consider issues such as encryption, public key infrastructure and two-factor authentication, as well as security accreditation reporting required by laws like the Federal Information Security Management Act (FISMA) and regulations such as the Federal Risk and Authorization Management Program (FedRAMP). He expects a sustained surge in government app development focused on mobile platforms.

"There's great opportunity in the government market," he says. "Everyone in the country is affected by all levels of government and there are still lots of problems to be solved."





Software Citizens

Government is often stereotyped as plodding and stifling innovation. If you know your history, though, you know much of what we do in IT actually started as government research projects. The Internet and World Wide Web owe their existence to public-sector organizations such as the Defense Advanced Research Projects Agency (DARPA) and the European Organization for Nuclear Research, also known as CERN.

Likewise, government-run services such as GPS and domain name registration yielded enormous economic benefit once these resources were opened up.

There's great value still locked up in the data government agencies collect, and key folks at the federal level know it. U.S. federal government CIO Steven VanRoekel and CTO Todd Park are spearheading the effort to challenge agencies to make more of their data accessible to both the public and application developers.

At howto.gov, you can see how government is training its agencies to create APIs, use cloud computing and create mobile applications. At data.gov, established in the early days of the current administration, you can find a central repository of government data sets, APIs, and even some apps developed by agencies and citizens.

The opportunities in Washington, D.C., have never been better for small or independent developers. In my role of leading business development for Microsoft's federal business, I work with the people and firms in the government space. I've gained some insight into what's getting traction in the market.

The emphasis of the Digital Government initiative is on citizen-facing apps. Agencies have already delivered 137 mobile apps and "citizen-developers" another 349. App development is facilitated by 295 published APIs across the full spectrum of government, from the Department of Agriculture to the White House.

There's a fair amount of variation across the APIs, so developers should understand SOAP and RESTful approaches. Ideally, there would be consistency in how agencies expose APIs, but given the vast nature of the federal government, it's unlikely that will happen. Just having the APIs in one place is a big win.

Good work has been done in the Digital Government space and there's more to come. One developer I spoke with hopes to create an app for "safe directions," combining standard driving directions from mapping services such as Bing and Google with historical traffic accident data to help drivers avoid risky routes. The challenge: traffic data isn't consistently captured by local government entities.

In fact, state and local governments likely hold a lot more useful data than the federal government. For example, a restaurant review site hoped to add health-inspection data to its reviews. The company piloted the feature in New York City, where it has good data from a single jurisdiction. In the D.C. metro area, however, multiple state and local jurisdictions handle the inspection data in different ways.

Efforts are underway to address this. The Open Government Data Initiative, found on GitHub at bit.ly/14M5tAq, is a cloud-based open data catalog for governments that want to give citizens access to data, enable developers to access the data via APIs, and streamline publishing of data directly from government systems. The goal is to lower the cost of sharing data and increase its value to society. There's already been uptake by early adopters like the European Union and the national governments of Estonia and Colombia.

Identity Crisis

The availability of data will surely improve over time, but challenges remain in the area of identity management. For government agencies, relying on usernames and passwords for authentication is hardly sustainable. Government agencies with citizen-facing missions don't want to manage end-user passwords any more than a small app developer does.

A solution may be in the offing. The Federal Cloud Credential Exchange (FCCX) aims to let government agencies trust ID credentials issued and authenticated by a third party. FCCX would let citizens use digital identities (including usernames/passwords) issued by their bank, for example, to log onto government Web sites. Conceptually it's not that different than what Facebook, Microsoft, Twitter and Google already do, but with a much higher degree of identity verification and authentication.

Associated with FCCX, which just entered a one-year pilot phase, is the concept of Trust Framework Providers. These are certified by the federal Identity, Credential and Access Management authority via the Trust Framework Adoption Process.

The government has already assessed and approved a number of identity providers such as Citibank, Entrust, DigiCert, VeriSign and even PayPal. That list is sure to grow as more citizen to government interactions become purely digital. You can find more information on this topic at idmanagement.gov.

The future for developers working in the government market is very bright. The government is embracing new ways of doing business and is much more open to small developers than they have been in the past.

I'm lucky to have a seat with a great view of this developing market, and I hope to see you in it soon! ■

GREG BATEMAN *is the senior director for Acquisition Programs, Policy and Strategy at Microsoft Federal, where his team is responsible for strategic business development in the federal market.*

Introducing RaptorXML®

The new, **hyper-fast XML and XBRL server** from the makers of XMLSpy®

- Ultra-high performance
- Ultra-low memory foot print
- Cross platform
- Optimized for parallel computing

Big Data trends and standards mandates are producing huge, ever increasing amounts of XML and XBRL data. Now, there is finally a modern, lightning-fast server to validate, process, transform, and query all of it: RaptorXML.

Standards Support:

XML 1.0, 1.1
XML Namespaces
XML Base
XInclude 1.0
XLink 1.0
XML Schema 1.0, 1.1
XPath 1.0, 2.0, 3.0
XSLT 1.0, 2.0, 3.0
XQuery 1.0, 3.0
Oasis Catalogs V1.1
XBRL 2.1
XBRL Dimensions
XBRL Formula 1.0
XBRL Functions
XBRL Definition Links

Part of the complete family of XML solutions from Altova

Any-to-any Data Mapping

XML, XSLT, and XBRL Development

Rendering & Report Generation

See the complete list of supported specifications and platforms, and download RaptorXML at

www.altova.com/raptorxml

BRUNO TERKALY AND
RICARDO VILLALOBOS

Choosing a Cloud Network for Government-Compliant Applications

There's already strong evidence that the government is moving to the cloud. The White House has committed to making government data on crime, public works and so forth open to the world, mandating its multiple branches to fully embrace the cloud, and even creating portals such as info.apps.gov, which describes itself as "a place where agencies can gather information about how cloud computing can create sustainable, more cost-effective IT services for the federal government." Other countries have implemented similar policies, such as the G-Cloud in the United Kingdom, the Seventh Framework Programme (FP7) for the European Union, and the e-Government initiative in Japan. Government agencies can clearly benefit from public cloud services for some specific scenarios, including the delivery of citizen services, the distribution of electronic content and the ability to collaborate remotely on common data.

Cloud Deployment Models

It's important to differentiate the cloud deployment models that government agencies around the world are following, because they determine how compute resources are provisioned, managed and consumed. There are basically two: public and private clouds. Microsoft defines the public cloud as a pool of compute and data resources that can be shared among multiple users, offering a pay-per-use economic model that clearly differentiates from the traditional purchase-in-advance approach to servers and storage mechanisms. Individual developers, companies, and organizations can self-provision IT infrastructure around the world by following simple steps, based on different usage patterns that usually present high variations in traffic or required compute power. Private clouds, on the other hand, are operated solely for one organization, may be managed by a third party, and are traditionally hosted either on-premises or isolated from other tenants in datacenters, providing a higher level of control and flexibility, but also requiring more maintenance and up-front investment.

According to the inaugural IDC Government Insights Report, "Perspective: Growth and Slight Contraction – Government Cloud Spending by U.S. Federal Agency," released July 15, 2013, spending on public clouds for government agencies is currently only about 10 percent of that for private cloud solutions, mostly because of security, privacy and compliance concerns. As a reference, federal private cloud services spending in the United States will reach $1.7 billion during FY2014, with an expectation it will reach $7.7 billion by FY2017. We believe the factors that motivate the adoption of private over public clouds will dissipate over time, as cloud vendors prove their efficacy in terms of privacy and data protection. It's a familiar trend in technology—trust takes time, despite the fact that government servers are frequently compromised by hackers.

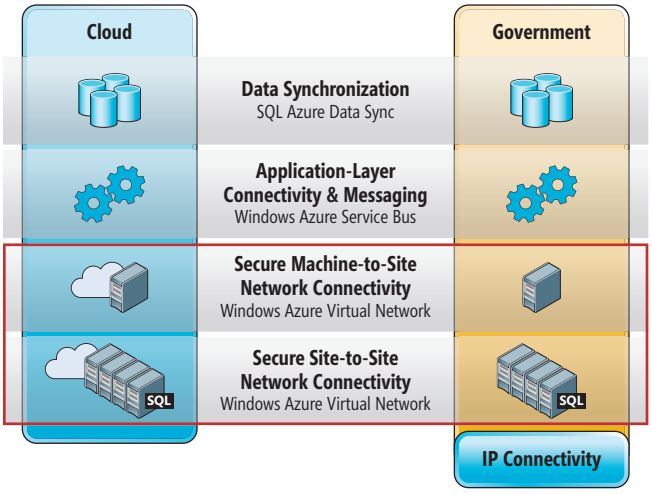
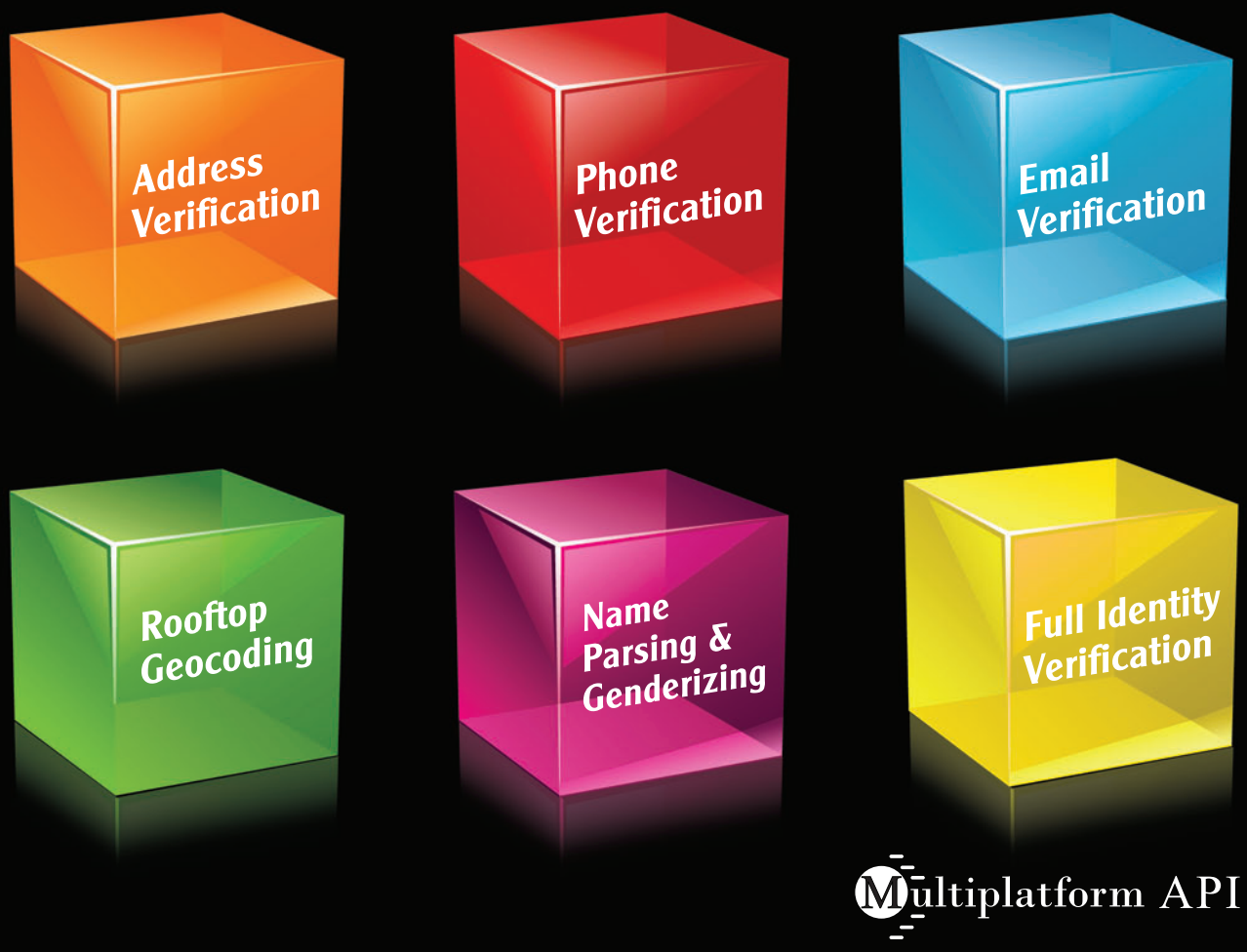


Figure 1 **Cross-Premises Connectivity Using Windows Azure**

> Microsoft defines the public cloud as a pool of compute and data resources that can be shared among multiple users.

Microsoft is committed to providing Windows Azure customers with detailed information about its security compliance programs to help them make their own regulatory assessments. Among the certifications that Windows Azure has obtained are the International Organization for Standardization/International Electrotechnical Commission (ISO/IEC) 27001:2005, the Statements on Standards for Attestation Engagement (SSAE) 16/International Standard on Assurance Engagements (ISAE) 3402 Attestation, and the Health Insurance Portability and Accountability Act (HIPAA) Business Associate Agreement (BAA). More information about security,

Building Blocks for Global Data Quality Success



A strong foundation for enterprise data starts with Melissa Data.

Our powerful, scalable data cleansing and integration tools help you profile, cleanse, consolidate, and enrich your contact data. Gain a better understanding of your customer, vendor and supplier data; improve deliverability; increase cost savings; and enhance your operational efficiencies with Melissa Data.

Advanced Functionalities:

- Verify, correct, and enhance addresses for 240+ countries
- Add lat/long coordinates to addresses all over the world
- Match and consolidate data to create the golden record
- Add missing contact data like phone/email information
- Integrate into many technologies with multiplatform capabilities

Thank you to all our government customers, including:

- U.S. Census Bureau
- Federal Bereau of Investigations
- Federal Aviation Administration
- Department of Veteran Affairs
- Florida DMV
- Maryland DMV

www.MelissaData.com
or call 1-800-MELISSA (635-4772)

MELISSA DATA®
Your Partner in Data Quality

privacy and compliance for Windows Azure can be found at the online trust center, windowsazure.com/trustcenter.

Other factors that will accelerate the adoption of the public cloud by government organizations are the available tools and infrastructure that enable the deployment of hybrid architectures (public and private), particularly for scenarios where some of the data resides on-premises, such as secured locations owned by government agencies. Windows Azure offers four ways today to enable these situations, as shown in **Figure 1**.

The first option, data synchronization, is directly related to solutions using Microsoft SQL Server as a service in the cloud (known as Windows Azure SQL Database), which can be synchronized with both on-premises or private databases for backup or replication purposes. To synchronize the data, sync groups—which define the databases, tables and columns—are created to specify synchronization schedules. Each sync group must have at least one SQL database instance that serves as the sync group hub in a hub-and-spoke topology. This is an easy way to maintain relational information deployed in multiple locations, either public or private.

The second option is the use of the Windows Azure Service Bus, which supports the implementation of messaging patterns to securely exchange information between distributed components of the solution. These endpoints are usually machines located in remote locations (behind private firewalls) or mobile personal devices. Different patterns are supported, including queues where receivers compete for a single message, or a publisher-subscriber model, where messages are sent to topics that distribute information to multiple receivers. More information and specific implementations of the Windows Azure Service Bus can be found in our December 2012 Windows Azure Insider column, "Windows Azure Service Bus: Messaging Patterns Using Sessions" (msdn.microsoft.com/magazine/jj863132).

The third and fourth options are based on a recent commercially released component, the Windows Azure Virtual Network, which offers two types of cross-premises VPN connections: site-to-site and point-to-site. A site-to-site VPN creates a secure connection between private facilities—such as a government-owned building—and resources in the cloud. Once the VPN connection is created, resources at both ends can communicate directly and securely, as shown in **Figure 2**. However, there's an extra requirement: the government office must provide an externally facing IPv4 IP address on a VPN device. The beauty of this site-to-site configuration is that it isn't necessary to individually configure each client computer that wants to be part of the VPN. Moreover, Cisco Systems Inc. has released a VPN device called the ASA Model 5505 for less than $300 that has the functionality of devices that often cost more than $1,000. Leveraging the Windows Azure Virtual Network service makes it a simple four-step process to create a VPN. This amazing combination of technologies has really democratized VPNs.

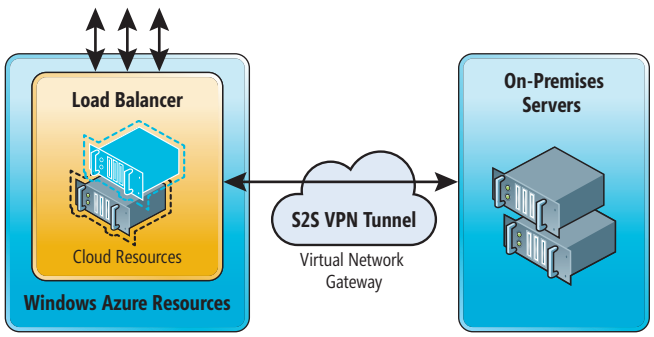


Figure 2 **Using Windows Azure Virtual Network VPN Functionality to Connect On-Premises Resources**

The point-to-site approach, in contrast, requires IT staff to individually configure each client computer, but with the benefit that no VPN device is needed. Instead, a VPN software client is installed on the individual computers. This makes sense for specific scenarios, but IT departments are not particularly fond of this solution because it requires more maintenance in the long term.

The Windows Azure portal makes it trivial to establish a VPN. Naturally, there's some homework to be done before the Windows Azure portal can be used. You need to know the details of your network and define subnets for the different machines that will be connecting to it. Once the requisite information is defined and collected, you can set up the VPN via the Windows Azure portal or by using Windows PowerShell/command-line interface (CLI) scripts.

> Typically, government agencies are more comfortable with the IaaS model due to the level of control it provides.

Here's what you need to know before using the Windows Azure portal:

- The location of the datacenter where the VPN service will be hosted.
- The name and IP address of the on-premises or cloud-based DNS server.
- The IP address of the government, on-premises VPN device.
- The various on-premises subnets that are in place.

The following steps, also shown in **Figure 3**, outline the basic process:

1. From the Windows Azure portal select Networks | Create a virtual network.
2. Provide the name and region where you want to host your VPN service. This step also includes potential affinity groups you might want to use.
3. Enter the name and IP address of your on-premises or cloud DNS server.
4. Specify the IP address of your VPN device (that is, the IP address of the Cisco VPN device, for example), as well as the address space of the on-premises network, including the address count. The usable address range is then calculated and displayed.
5. Add the necessary subnets pertaining to your on-premises network.

After creating your Windows Azure Virtual Network, you can use the detailed, step-by-step directions described at bit.ly/KYIkvK

Windows. Web. Mobile.
Your next great app starts here.

DevExpress .NET controls, frameworks and libraries were built with you in mind. Built for those who demand the highest quality and expect the best performance... for those who require reliable tools engineered to meet today's needs and address tomorrow's requirements.

Experience the DevExpress difference today and download your free 30-day trial and let's build great apps, together.

WinForms · ASP.NET · WPF · Silverlight · Windows 8 XAML · HTML JS · Reporting · DevExtreme Mobile

Learn more and download your free trial
devexpress.com/try

DevExpress™

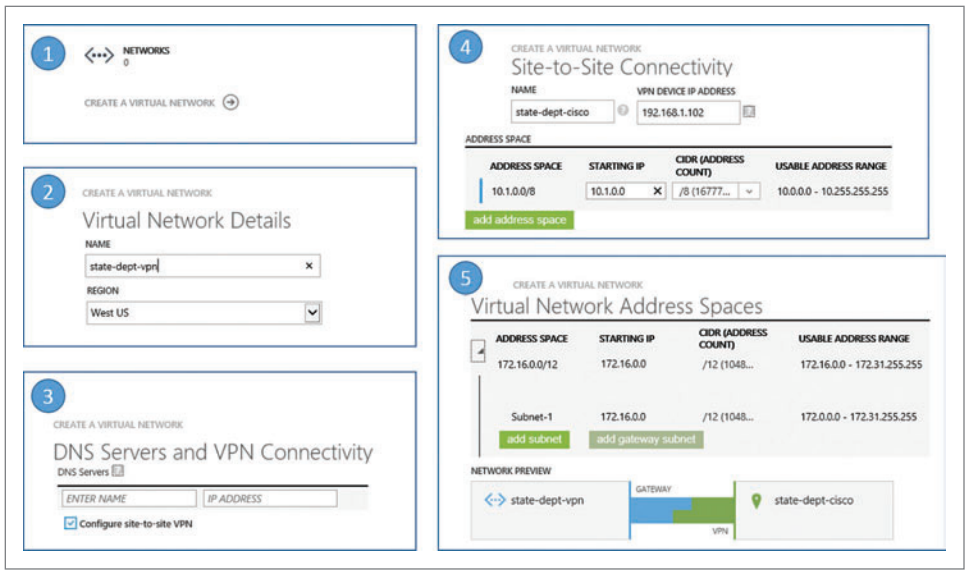


Figure 3 **Creating a VPN**

to configure the virtual network gateway in order to create your site-to-site VPN. There will be VPN device-specific configuration scripts you'll need to get from the Windows Azure portal to simplify administration. You'll also need to indicate a gateway IP address and a shared key.

Cloud Service Models

Another way of differentiating the cloud is by service model: Software as a Service (SaaS), Platform as a Service (PaaS) or Infrastructure as a Service (IaaS). The models simply define how much of the technology stack is delegated to the public cloud vendor (see **Figure 4**).

In the SaaS model, the consumer simply accesses an application hosted in the cloud through a Web browser or thin client, as is the case with social networks such as Facebook and Twitter, productivity tools such as Office 365 and communication/collaboration hubs such as outlook.com. The PaaS model is used mostly by developers, who can provision compute resources in the cloud using different programming languages and tools provided by the cloud vendor. Finally, IaaS allows IT pros to deploy, manage, and control fundamental computing resources such as virtual machines (VMs) and networks. IaaS requires more maintenance time and planning, but it yields more flexibility and control over the lower layers of the technology stack.

Typically, government agencies are more comfortable with the IaaS model due to the level of control it provides. According to the IDC Government Insights report mentioned earlier, between now and 2017, IaaS spending will grow to $5.4 billion, SaaS will grow to $2.4 billion, and PaaS will grow to $1.1 billion for U.S. government agencies.

One of the features of the Windows Azure Virtual Network is that, in addition to the combination of public and private cloud deployments, it allows the combining of multiple service models, resulting in rich architectures that can take advantage of different platforms without compromising control or flexibility. As an example, Web portals can be deployed using the cloud services (PaaS) component of Windows Azure, while the database engine (which might require customized settings or configuration) is deployed to a VM (IaaS), keeping the communication between them exclusively and securely internal to the datacenter. This means there's no need to expose endpoints to the Web or to open external endpoints enabling the solution on the Web server to extract, update or remove information from the database (**Figure 5**). This particular feature, which provides high levels of connectivity without compromising security, is one of the biggest differentiators of the Windows Azure platform compared to other cloud platforms.

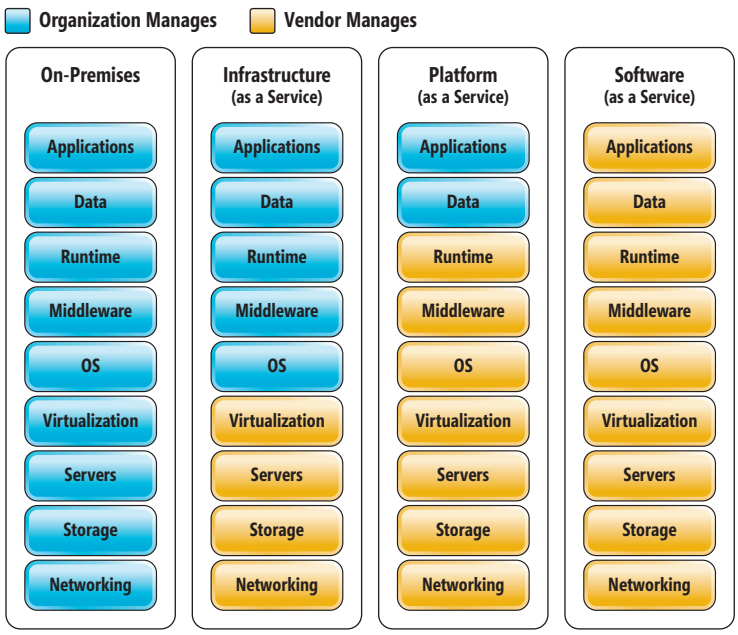


Figure 4 **Cloud Service Models**

The process of provisioning a hybrid service model (IaaS and PaaS) with Windows Azure is similar to the one mentioned earlier for creating VPN connections, requiring the definition of a virtual network, and then the addition of VMs (IaaS) and cloud services (PaaS).

For the PaaS components, the configuration file needs to include information about the virtual network. If you aren't familiar with the deployment process for PaaS in Windows Azure, you'll want to read the official documentation and concepts at bit.ly/17YKcym. **Figure 6** lists the basic network configuration schema for cloud services.

You'll find that many of the virtual network configuration elements we mentioned that need to be defined

BEST SELLER

DevExpress Universal Suite | from $1,924.99

DevExpress

Over 400 WinForms, ASP.NET, WPF, Silverlight and Windows 8 tools & controls.

- Use your existing codebase to develop touch-enabled apps across platforms
- Includes cross-platform data visualization dashboard and report server
- Ships with new DevExpress controls including Spreadsheet and Map Control
- Code, debug and refactor with CodeRush for Visual Studio
- Includes new unified DevExpress Application Template Gallery

BEST SELLER

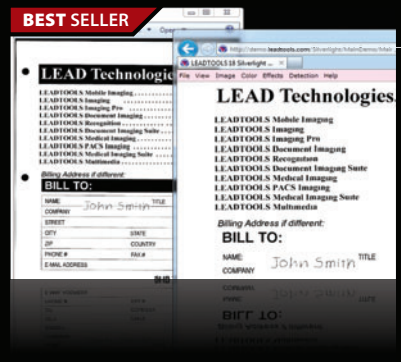
ComponentOne Studio Enterprise 2013 v2 | from $1,293.18

ComponentOne
a division of GrapeCity

.NET Tools for the Professional Developer: Windows, HTML5/Web, and XAML.

- Hundreds of UI controls for all .NET platforms including grids, charts, reports and schedulers
- Visual Studio 2013 and Windows 8.1 support
- 40+ UI widgets built with HTML5, jQuery, CSS3, and SVG
- New TouchToolkit for touch enabling WinForms apps
- Royalty-free deployment and distribution

BEST SELLER



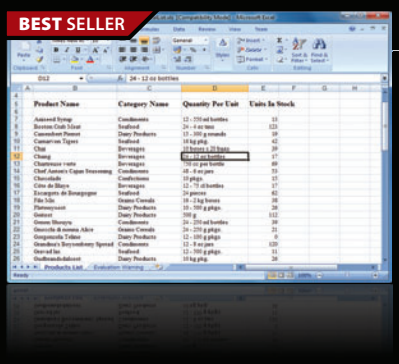
LEADTOOLS Document Imaging SDKs V17.5 | from $2,545.75

LEAD TECHNOLOGIES

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

- Comprehensive document image cleanup, preprocessing, viewer controls and annotations
- Fast and accurate OCR, ICR and Forms Recognition with multi-threading support
- PDF & PDF/A Read / Write / Extract / View / Edit
- Barcode Detect / Read / Write for UPC, EAN, Code 128, Data Matrix, QR Code, PDF417
- Native WinRT Libraries for Windows Store & Zero-Footprint HTML5/JavaScript Controls

BEST SELLER



Aspose.Total for .NET | from $2,124.15

ASPOSE
Your File Format Experts

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 document management in .NET applications
- Common uses also include mail merging, adding barcodes to documents, building dynamic reports on the fly and extracting text from PDF files

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

US Government Sales:
(888) 850-9966
www.componentsource.com

GSA Schedule Contract GS-35F-0188R

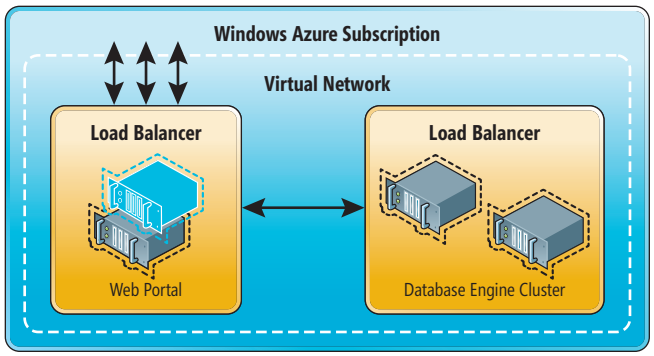


Figure 5 **Combining PaaS and IaaS Deployments in Windows Azure Using Virtual Networks**

in advance for VPN connections are also included in the cloud service configuration file schema. Keep in mind that if the cloud service role is already running on Windows Azure, it will need to be redeployed with the new configuration file information before it can be added to the virtual network. Detailed descriptions for each of the schema elements can be found at bit.ly/162xahL.

Adding VMs (IaaS) to virtual networks can be accomplished through the Windows Azure Management Portal, or by using Windows PowerShell or CLI scripts. The process is fully described on the Windows Azure page, "Add a Virtual Machine to a Virtual Network," at bit.ly/KHUixg. Similar to cloud service roles, VMs already deployed need to be stopped and re-provisioned before they can be added to virtual networks.

Figure 6 **Network Configuration Schema for Adding Cloud Services to Virtual Networks**

```
<ServiceConfiguration ...>
  <NetworkConfiguration>
  <Dns>
    <DnsServers>
      <DnsServer name="name1" IPAddress="IPAddress1" />
      <DnsServer name="name2" IPAddress="IPAddress2" />
      <DnsServer name="name3" IPAddress="IPAddress3" />
    </DnsServers>
  </Dns>
  <VirtualNetworkSite name="name1"/>
  <AddressAssignments>
    <InstanceAddress roleName="roleName1">
      <Subnets>
        <Subnet name="name1" />
        <Subnet name="name2" />
        <Subnet name="name3" />
      </Subnets>
    </InstanceAddress>
    <InstanceAddress roleName="roleName2">
      <Subnets>
        <Subnet name="name4" />
        <Subnet name="name5" />
        <Subnet name="name6" />
      </Subnets>
    </InstanceAddress>
    <InstanceAddress roleName="roleName3">
      <Subnets>
        <Subnet name="name7" />
        <Subnet name="name8" />
        <Subnet name="name9" />
      </Subnets>
    </InstanceAddress>
    </AddressAssignments>
  </NetworkConfiguration>
</ServiceConfiguration>
```

Combining PaaS and IaaS models in the same solution allows you to choose the right model for each architecture component. Government agencies using IaaS today as their primary deployment model can start identifying areas that can benefit from PaaS without losing control over specific servers in their architectures. We see PaaS growing in significance as developers learn the advantages of this deployment model—easy scaling, automatic updates and rolling upgrades, among others.

Hybrid cloud networks—both from a deployment and service perspective—provide different advantages to government agencies designing solutions that need to follow specific privacy, security and compliance policies.

Wrapping Up

Hybrid cloud networks—both from a deployment and service perspective—provide different advantages to government agencies designing solutions that need to follow specific privacy, security and compliance policies. Sensitive compute and storage resources can be kept on-premises or in private clouds, while being securely connected to less-critical components running or stored in public clouds, such as Windows Azure. At the same time, government applications that are based only on IaaS can start taking advantage of other service models such as PaaS, which require less maintenance and support time. ■

Bruno Terkaly *is a developer evangelist for Microsoft. His depth of knowledge comes from years of experience in the field, writing code using a multitude of platforms, languages, frameworks, SDKs, libraries and APIs. He spends time writing code, blogging and giving live presentations on building cloud-based applications, specifically using the Windows Azure platform. You can read his blog at blogs.msdn.com/b/brunoterkaly.*

Ricardo Villalobos *is a seasoned software architect with more than 15 years of experience designing and creating applications for companies in the supply chain management industry. Holding different technical certifications, as well as a master's degree in business administration from the University of Dallas, he works as a platform evangelist in the Globally Engaged Partners DPE group for Microsoft. You can read his blog at blog.ricardovillalobos.com.*

Terkaly and Villalobos jointly present at large industry conferences. They encourage readers of Windows Azure Insider to contact them for availability. Terkaly can be reached at bterkaly@microsoft.com and Villalobos can be reached at Ricardo.Villalobos@microsoft.com.

Thanks to the following technical expert for reviewing this article: *Dennis Velázquez (Microsoft)*

TX TEXT CONTROL®
word processing components

Creating a report is as easy as writing a letter

- Reuse MS Word documents as your reporting templates
- Create encrypted and print-ready Adobe PDF and PDF/A
- Royalty-free WYSIWYG template designer
- Powerful and dynamic 2D/3D charting support
- Easy database connections and master-detail nested blocks
- 1D/2D barcode support including QRCode, IntelligentMail, EAN

www.textcontrol.com/reporting

txtextcontrol

US: +1 855-533-TEXT
EU: +49 421 427 067-10

Visual Studio Microsoft Partner

Reporting | Rich Text Editing | Spell Checking | Barcodes | PDF Reflow

JAMES MCCAFFREY

Implementing the National Institute of Standards and Technology Tests of Randomness Using C#

There's a long history of support and cooperation between U.S. government organizations and the software development community. This article presents C# code that implements tests of randomness devised by the National Institute of Standards and Technology (NIST). In addition to providing a practical and useful addition to your programming toolset, I hope this article will persuade you that using .NET technologies to implement algorithms created by federal organizations can be significantly easier than using older technologies.

The best way to see where this article is headed is to take a look at the screenshot of a demo program in **Figure 1**. The demo program begins by creating and displaying a sequence of 52 binary tokens represented by 0s and 1s:

```
1100 0011 1010 1110 0000 1111 0000 1111 0000 1111 0000 1111 0000
```

Imagine that each token represents the color of a playing card (0 for black spades and clubs, and 1 for red hearts and diamonds), and the sequence shown is the order in which cards were dealt to you by gambling-simulation software. Does the sequence of 0s and 1s appear random, as it should be, or does the sequence look suspicious? Notice the first 16 cards seem to be rather unexceptional, but the remaining 36 cards appear to have a pattern. Did this happen by chance, or is the sequence statistically suspicious?

The C# demo program runs three tests that were devised by NIST. The first test checks the sequence for the overall frequencies of 0s and 1s. In this case, there are 25 1s and 27 0s, and the frequency test concludes that this is not suspicious. The second test examines the sequence by dividing it into six blocks of eight tokens, with the last four tokens not used. The block test also concludes there's nothing suspicious about the pattern. The third NIST test checks the sequence for runs, which are sub-sequences in which all the tokens are the same. In this case there are 16 runs. The third test concludes that the number of runs in the input pattern is highly unlikely to have occurred if the cards were dealt randomly.

This article assumes you have at least intermediate-level skills with .NET technologies, but doesn't assume you're very knowledgeable about statistics. The demo program is coded in C# but you should have no trouble refactoring the demo to other .NET languages such as Visual Basic .NET or Windows PowerShell. The demo code has most error checking removed to keep the main ideas clear. The entire demo source code is available at archive.msdn.microsoft.com/mag201310gTestRun.

The reference document for this article is "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications," NIST Special Publication 800-22, Revision 1a. I'll refer to the document as the Test Suite. The document is available from csrc.nist.gov.

The Test Suite describes a total of 15 tests for randomness. This article presents C# implementations of the first three tests. The Test Suite was originally created before the existence of .NET technologies, so NIST provides example C language code. I'm a big fan of the C language, but C wasn't really designed for tasks such as testing for randomness. The C download from NIST includes 10 .h files and 22 .c files, and each of the 32 files can have several functions. Even though the sample code is clearly explained, in my opinion working with a single C# file is much easier than dealing with 32 C files.

Overall Program Structure

The overall program structure of the demo, with a few minor edits, is presented in **Figure 2**.

To create the demo, I launched Visual Studio. The demo has no significant .NET dependencies and any version of Visual Studio should work. I created a new C# console application project named NistRandomness. After the template code loaded, in the Solution Explorer window I renamed file Program.cs to the more descriptive NistProgram.cs, and Visual Studio automatically renamed class Program for me. At the top of the template-generated code, I deleted all references to namespaces except for the ones to System and Collections.

In addition to the Main method, the demo has seven static methods that implement the three tests and helper routines to store a binary sequence, plus a GammaFunction class that houses math routines used by the BlockTest method.

The demo sets up a sequence of binary tokens using a string approach:

```
string bitString = "1100 0011 ...";
BitArray bitArray = MakeBitArray(bitString);
```

In most cases, you'd be testing a sequence generated by some other system, such as a cryptographic method, or perhaps even a gambling

Code download available at archive.msdn.microsoft.com/mag201310gTestRun.

Build native mobile apps in C# and Visual Studio

Cross-Platform Development Redefined

Fast time to market
Multiply your device reach in a fraction of the time thanks to code-sharing and re-use.

No compromises
Anything that can be done in Objective-C and Java can be done with Xamarin.

Instant productivity
Leverage your existing C# teams, tools and code for mobile development.

Join over **400,000** developers and thousands of businesses who are delivering mission-critical customer and enterprise apps with Xamarin.

Xamarin

simulation. In general, you shouldn't have too much difficulty writing code that transforms your sequence into a string of 0s and 1s.

Calling all 15 of the NIST tests follows a consistent pattern where the input sequence is passed to a test method and the test method returns a probability value between 0 and 1. For example:

```
double pFreq = FrequencyTest(bitArray);
```

The probability represents the likelihood that the input pattern could've happened by chance if the pattern was in fact generated randomly. So very small values—generally those less than 0.05 or 0.01—suggest the input pattern isn't random. The Test Suite recommends using a 0.01 significance level, but I recommend 0.05 in most cases.

The BitArray Class

The .NET BitArray class is well-suited for efficiently storing a sequence of binary tokens. Method MakeBitArray is defined in **Figure 3**.

A BitArray object is a bit unusual (no pun intended) in the sense that even though behind the scenes the contents are stored as bits, an object's interface is exposed using Boolean true/false values. For example:

```
bool[] vals = new bool[] { true, false, true, false };
BitArray ba = new BitArray(vals);
```

This approach is extremely tedious for long input sequences. Method MakeBitArray accepts a string of 0s and 1s, and allows spaces for readability.

Method ShowBitArray accepts a BitArray object to display, and parameters to control where spaces and line breaks are placed:

```
static void ShowBitArray(BitArray bitArray, int blockSize,
  int lineSize)
{
  for (int i = 0; i < bitArray.Length; ++i) {
    if (i > 0 && i % blockSize == 0) Console.Write(" ");
    if (i > 0 && i % lineSize == 0) Console.WriteLine("");

    if (bitArray[i] == false) Console.Write("0");
    else Console.Write("1");
  }
  Console.WriteLine("");
}
```

The Frequency Test

Consider the pattern 0100 0000 0000 0010. The pattern doesn't appear to be random because there are too many 0s. The most fundamental test for randomness is a frequency test. If a pattern is generated randomly, you'd expect the number of 0s and 1s to be roughly the same. Too many 0s or too many 1s suggest non-randomness. Method FrequencyTest computes a sum where 0s are encoded as -1 values and 1s are encoded as +1 values. If the sum is 0, there are equal numbers of 0s and 1s, but a sum different from 0, either very negative or very positive, means too many 0s or 1s. In the preceding example pattern, the sum would be -12.

The NIST Test Suite document is, in my opinion, a fantastic example of exactly how software documentation should be written: clear, concise and with excellent examples. The document explains exactly how the Frequency test works on page 2-2, and I had no problem implementing it:

```
static double FrequencyTest(BitArray bitArray)
{
  double sum = 0;
  for (int i = 0; i < bitArray.Length; ++i) {
    if (bitArray[i] == false) sum = sum - 1;
    else sum = sum + 1;
  }
  double testStat = Math.Abs(sum) / Math.Sqrt(bitArray.Length);
  double rootTwo = 1.414213562373095;
  double pValue = ErrorFunctionComplement(testStat / rootTwo);
  return pValue;
}
```

The test statistic is the absolute value of the sum divided by the square root of the number of binary tokens. You can consider this a magic equation. The trickiest part of the Frequency test is the computation of the return p-value using method ErrorFunctionComplement. In math, there's an Error Function that has a specific definition. The complement of the Error Function is just 1.0 minus the values of the Error Function. The demo defines method ErrorFunctionComplement as:

```
static double ErrorFunctionComplement(double x)
{
  return 1 - ErrorFunction(x);
}
```

where:

```
static double ErrorFunction(double x)
{
  double p = 0.3275911;
  double a1 = 0.254829592;
  double a2 = -0.284496736;
  double a3 = 1.421413741;
  double a4 = -1.453152027;
  double a5 = 1.061405429;
  double t = 1.0 / (1.0 + p * x);
  double err = 1.0 - (((((a5 * t + a4) * t) +
    a3) * t + a2) * t + a1) * t * Math.Exp(-x * x);
  return err;
}
```

The Error Function is closely related to the Standard Normal Distribution (the bell-shaped curve) and can be used to determine how likely or unlikely a statistical event is. Method ErrorFunction uses one of many approximation equations to the true function.

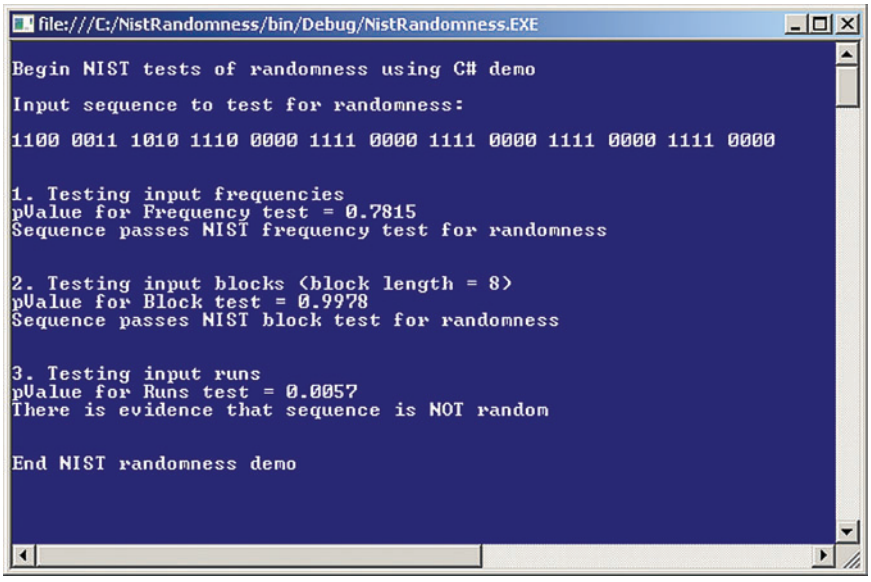
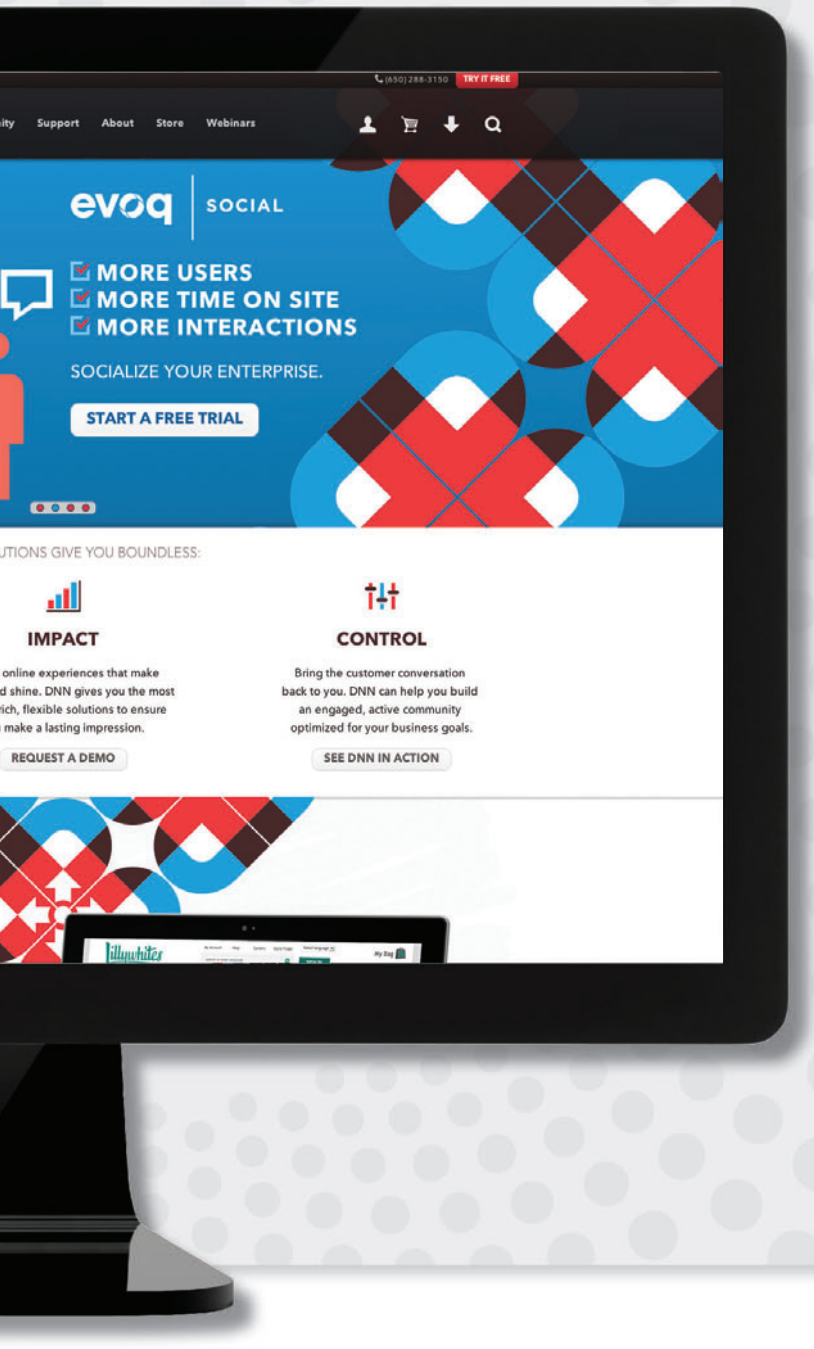


Figure 1 **National Institute of Standards and Technology Tests for Randomness Demo**



The BEST Web Content Management Solutions Come From DNN.

- Low Initial Investment
- Minimal Ongoing Expenses
- Future Proof Extensibility

Rated 100/100 by Infotech.

Info-Tech Research Group rated DNN the **"Best Overall Value"** in its 2013 Web Content Management Vendor Landscape.

VENDOR LANDSCAPE AWARD
CHAMPION
INFO~TECH research group

VENDOR LANDSCAPE AWARD
BEST OVERALL VALUE
INFO~TECH research group

For a free trial, visit www.dnnsoftware.com/trial

DNN

Figure 2 **Demo Program Structure**

```
using System;
using System.Collections;

namespace NistRandomness
{
  class NistProgram
  {
    static void Main(string[] args)
    {
      try
      {
        Console.WriteLine("Begin NIST tests of randomness using C# demo\n");

        string bitString = "1100 0011 1010 1110 0000 1111" +
          " 0000 1111 0000 1111 0000 1111 0000";
        BitArray bitArray = MakeBitArray(bitString);
        Console.WriteLine("Input sequence to test for randomness: \n");
        ShowBitArray(bitArray, 4, 52);

        Console.WriteLine("1. Testing input frequencies");
        double pFreq = FrequencyTest(bitArray);
        Console.WriteLine("pValue for Frequency test = " + pFreq.ToString("F4"));
        if (pFreq < 0.01)
          Console.WriteLine("There is evidence that sequence is NOT random");
        else
          Console.WriteLine("Sequence passes NIST frequency test for randomness");

        int blockLength = 8;
        Console.WriteLine("2. Testing input blocks (block length = " +
          blockLength + ")");
        double pBlock = BlockTest(bitArray, blockLength);
        Console.WriteLine("pValue for Block test = " + pBlock.ToString("F4"));
        if (pBlock < 0.01)
          Console.WriteLine("There is evidence that sequence is NOT random");
        else
          Console.WriteLine("Sequence passes NIST block test for randomness");

        Console.WriteLine("3. Testing input runs");
        double pRuns = RunsTest(bitArray);
        Console.WriteLine("pValue for Runs test = " + pRuns.ToString("F4"));
        if (pRuns < 0.01)
          Console.WriteLine("There is evidence that sequence is NOT random");
        else
          Console.WriteLine("Sequence passes NIST runs test for randomness");

        Console.WriteLine("End NIST randomness demo\n");
        Console.ReadLine();
      }
      catch (Exception ex)
      {
        Console.WriteLine(ex.Message);
        Console.ReadLine();
      }
    } // Main

    static BitArray MakeBitArray(string bitString) { . . }
    static void ShowBitArray(BitArray bitArray, int blockSize,
      int lineSize) { . . }

    static double FrequencyTest(BitArray bitArray)
    static double ErrorFunction(double x) { . . }
    static double ErrorFunctionComplementary(double x) { . . }

    static double BlockTest(BitArray bitArray, int blockLength) { . . }

    static double RunsTest(BitArray bitArray) { . . }
  }

  public class GammaFunctions { . . }
} // ns
```

Here I use equation 7.1.26, which is given in the landmark reference book, "Handbook of Mathematical Functions" (Dover Publications, 1965), by Abramowitz and Stegun. The book is often just called "A&S." This book is also a product of NIST and is indispensable if you work with numerical programming. "A&S" is freely available online at dlmf.nist.gov.

The Block Test

Consider the pattern 00000000 11111111. This pattern would pass the Frequency test because there are equal numbers of 0s and 1s, but clearly the pattern looks suspicious. The Block test is designed to address this type of non-randomness. The Block test divides a pattern into blocks and examines the number of 1s in each block. A random pattern would be expected to have about 50 percent 1s in every block. The Test Suite explains the algorithm for the Block test in detail on p. 2-4.

Figure 3 **The MakeBitArray Method**

```
static BitArray MakeBitArray(string bitString)
{
  int size = 0;
  for (int i = 0; i < bitString.Length; ++i)
    if (bitString[i] != ' ') ++size;

  BitArray result = new BitArray(size);
  int k = 0; // ptr into result
  for (int i = 0; i < bitString.Length; ++i) {
    if (bitString[i] == ' ') continue;
    if (bitString[i] == '1') result[k] = true;
    else result[k] = false;
    ++k;
  }
  return result;
}
```

Briefly, the Block test accepts a block length parameter that's the number of bits per block. From this, the number of blocks can be computed. Next, the Block test computes the proportion of 1s in each block and then uses a magic equation to compute a chi-squared test statistic. The code for method BlockTest is presented in **Figure 4**. An image that shows how the Block test works is shown in **Figure 5**.

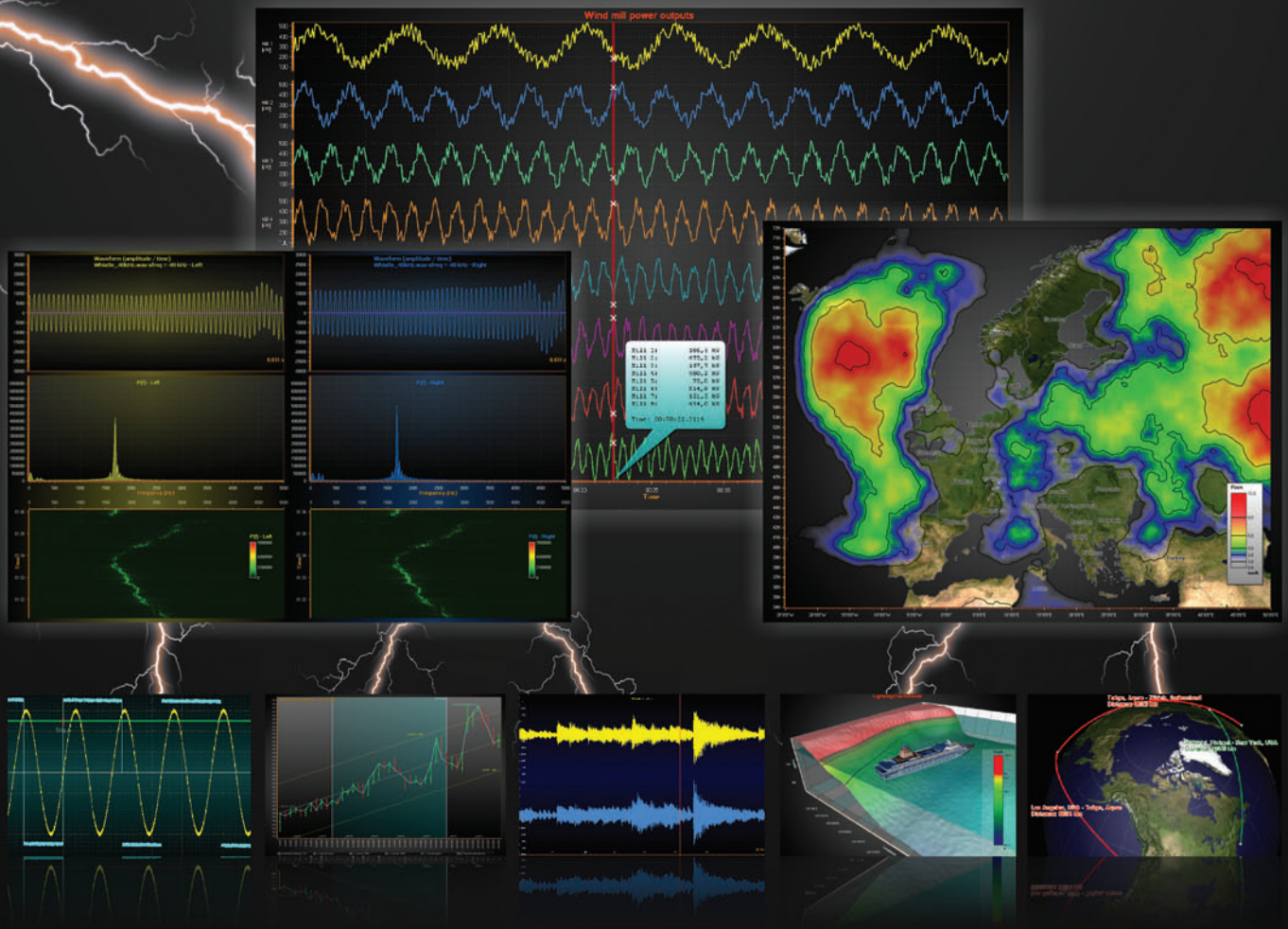
Figure 4 **The BlockTest Method**

```
static double BlockTest(BitArray bitArray, int blockLength)
{
  int numBlocks = bitArray.Length / blockLength; // 'N'

  double[] proportions = new double[numBlocks];
  int k = 0; // ptr into bitArray
  for (int block = 0; block < numBlocks; ++block)
  {
    int countOnes = 0;
    for (int i = 0; i < blockLength; ++i)
    {
      if (bitArray[k++] == true)
        ++countOnes;
    }
    proportions[block] = (countOnes * 1.0) / blockLength;
  }

  double summ = 0.0;
  for (int block = 0; block < numBlocks; ++block)
    summ = summ + (proportions[block] - 0.5) *
     (proportions[block] - 0.5);

  double chiSquared = 4 * blockLength * summ; // magic

  double a = numBlocks / 2.0;
  double x = chiSquared / 2.0;
  double pValue = GammaFunctions.GammaUpper(a, x);
  return pValue;
}
```

ULTIMATE CHARTING POWER

LightningChart

The fastest rendering data visualization components for WPF and WinForms



- **Fully DirectX accelerated**
- **Superior 2D and 3D rendering performance**
- **Very extensive property sets**
- **Optimized for real-time data monitoring**
- **Supports gigantic data sets**
- **Professional, friendly and fast customer support**
- **Compatible with Visual Studio 2005...2012**

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.

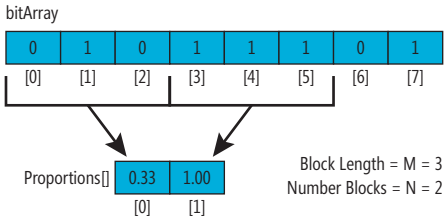
FREE TRIAL

Download a free 30-day evaluation from

www.LightningChart.com

Arction

Pioneers of high-performance data visualization



chi-squared = 4 * M * [$(0.33 - 0.5)^2 + (1.00 - 0.5)^2$] = 12 * [0.028 + 0.250] = 3.33

p-value = GammaQ(N/2, chi-squared/2) = GammaQ(1.0, 1.67) = 0.1889

Figure 5 **The Block Test Algorithm**

By far the most difficult part of the Block test—and of the entire Test Suite—is the computation of the return p-value using function GammaUpper.

Gamma Functions

It's quite likely you've never encountered Gamma functions before, but they're a cornerstone of numerical programming. There's one primary Gamma function and several closely related functions. The Block test uses a Gamma function that's, unfortunately, called by several different names, including the "incomplete upper Gamma function," "GammaQ," "upper Gamma complement" and "igamc."

The primary Gamma function essentially extends the notion of factorial to real values. Gamma functions can be used to determine the likelihood of a complex statistical event. The demo program places five Gamma functions into a class named GammaFunctions. These functions could've been placed in the demo program as standalone static functions just like ErrorFunction and ErrorFunctionComplement, but housing the Gamma functions together makes them a bit more manageable. Here's the structure of the GammaFunctions class:

```
public class GammaFunctions
{
  public static double GammaLower(double a, double x) { . . }
  public static double GammaUpper(double a, double x) { . . }
  private static double LogGamma(double x) { . . }
  private static double GammaLowerSer(double a, double x) { . . }
  private static double GammaUpperCont(double a, double x) { . . }
}
```

Bear with me for a moment. The one gamma function that's directly called by the Block test is GammaUpper. GammaUpper calls private helper functions GammaLowerSer and GammaUpperCont.

Figure 6 **The LogGamma Function**

```
public static double LogGamma(double x)
{
  double[] coef = new double[6] { 76.18009172947146, -86.50532032941677,
    24.01409824083091, -1.231739572450155,
    0.1208650973866179E-2, -0.5395239384953E-5 };
  double LogSqrtTwoPi = 0.9189385332046727 4178;
  double denom = x + 1;
  double y = x + 5.5;
  double series = 1.000000000190015;
  for (int i = 0; i < 6; ++i)
  {
    series += coef[i] / denom;
    denom += 1.0;
  }
  return (LogSqrtTwoPi + (x + 0.5) * Math.Log(y) -
    y + Math.Log(series / x));
}
```

Both helper functions call another helper function LogGamma. Class GammaFunctions also implements a function GammaLower that isn't used by the Block test.

The relationships among these Gamma functions can drive you batty if you're new to them, but you don't need to understand these relationships to effectively use the demo code. That said, the LogGamma function computes the log of the primary Gamma function. The actual primary Gamma function is rarely computed because it overflows for even moderate input values. Function LogGamma is presented in **Figure 6**. The LogGamma function has been studied for decades. The approximation used in the demo employs what's called the Lanczos algorithm.

Helper functions GammaLowerSer and GammaUpperCont are presented in **Figure 7**. The versions used here are derived from the well-known book, "Numerical Recipes in C" (Cambridge University Press, 2007), sometimes abbreviated "NR," which in turn uses the algorithms in "A&S." To be honest, I'm not a fan of the coding style used in "NR," but I've used most of the same, rather cryptic "NR" variable names so you can exploit "NR" as a resource. The second edition of "NR" is freely available online at bit.ly/p5gRye.

In some situations you might need to modify the code for GammaLowerSer and GammaUpperCont. Both functions are iterative approximations and stop when a maximum number of iterations (100) is reached or when an error threshold (3.0E-7 = 0.0000003) is achieved. If your Block test throws an exception, you can increase the max iterations, or increase the value of the error threshold, or both.

With the three helper functions in place, the GammaUpper function is very short:

```
public static double GammaUpper(double a, double x)
{
  // Incomplete Gamma 'Q' (upper)
  if (x < 0.0 || a <= 0.0)
    throw new Exception("Bad args in GammaUpper");
  if (x < a + 1)
    return 1.0 - GammaLowerSer(a, x); // Indirect is faster
  else
    return GammaUpperCont(a, x);
}
```

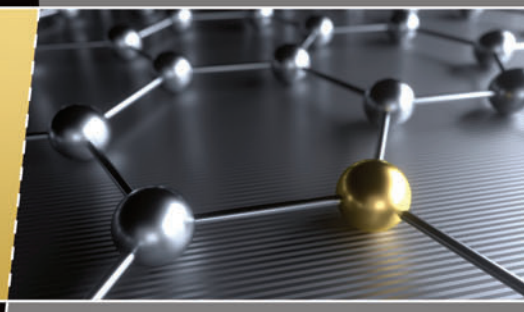
It's not immediately obvious what's going on here. It turns out that GammaLowerSer and GammaUpperCont are really the same functions in the sense that the value of either of them is just 1.0 minus the value of the other. In other words, if you know one you can easily compute the other. So why two versions? As it turns out, there are two techniques to estimate an incomplete Gamma function: by a series expansion or by a continuing fraction. The series expansion is more accurate and quicker when the value of input parameter x is less than the value of input parameter a+1. The continued fraction version is more accurate and quicker otherwise.

Although it isn't used by the Block test, the demo program implements the incomplete lower Gamma function (aka GammaP) as GammaLower:

```
public static double GammaLower(double a, double x)
{
  // Incomplete Gamma 'P' (lower) aka 'igam'
  if (x < 0.0 || a <= 0.0)
    throw new Exception("Bad args in GammaLower");
  if (x < a + 1)
    return GammaLowerSer(a, x); // No surprise
  else
    return 1.0 - GammaUpperCont(a, x); // Indirectly is faster
}
```

Empower Your Customers

New! v5.0



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

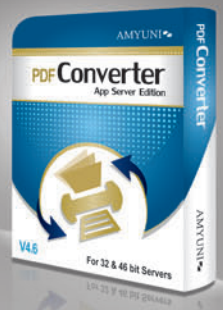
Complete Suite of Accurate PDF Components

- 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 certified PDF Converter printer driver
- Export PDF documents into other formats such as JPeg, PNG, XAML or HTML5

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 and Windows 8
- 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

The Runs Test

Consider the pattern 0101 0101 0101 0101. This pattern would pass the Frequency test because there are equal numbers of 0s and 1s. The pattern would also likely pass the Block test because each block would have roughly 50 percent of 0 bits and 50 percent of 1 bits (depending on whether the block size is even or odd). But, pretty clearly, the pattern doesn't appear random. The Runs test catches patterns like this. A run is a sequence where consecutive bit tokens are the same. For example, the pattern 00100011 has four runs: 00, 1, 000 and 11. If a pattern is randomly generated, it's possible to compute the expected number of runs.

The Runs test is presented in **Figure 8**. The test first computes the proportion of 1s in the input pattern. Then the number of runs is computed using the idea that if you walk-through a pattern, a bit at a time, every time you see a change in bit value you've encountered the start of a new run.

The magic test statistic is computed using the overall proportion of 1s and the number of runs. The return p-value is computed using the complement of the Error Function, just as in the Frequency test.

Figure 7 **Incomplete Gamma Function Helpers**

```
private static double GammaLowerSer(double a, double x)
{
  // Incomplete GammaLower (computed by series expansion)
  if ( x < 0.0)
    throw new Exception("x param less than 0.0 in GammaLowerSer");

  double gln = LogGamma(a);
  double ap = a;
  double del = 1.0 / a;
  double sum = del;
  for (int n = 1; n <= 100; ++n)
  {
    ++ap;
    del *= x / ap;
    sum += del;
    if (Math.Abs(del) < Math.Abs(sum) * 3.0E-7) // Close enough?
      return sum * Math.Exp(-x + a * Math.Log(x) - gln);
  }
  throw new Exception("Unable to compute GammaLowerSer " +
    "to desired accuracy");
}

private static double GammaUpperCont(double a, double x)
{
  // Incomplete GammaUpper computed by continuing fraction
  if (x < 0.0)
    throw new Exception("x param less than 0.0 in GammaUpperCont");
  double gln = LogGamma(a);
  double b = x + 1.0 - a;
  double c = 1.0 / 1.0E-30; // Div by close to double.MinValue
  double d = 1.0 / b;
  double h = d;
  for (int i = 1; i <= 100; ++i)
  {
    double an = -i * (i - a);
    b += 2.0;
    d = an * d + b;
    if (Math.Abs(d) < 1.0E-30) d = 1.0E-30; // Got too small?
    c = b + an / c;
    if (Math.Abs(c) < 1.0E-30) c = 1.0E-30;
    d = 1.0 / d;
    double del = d * c;
    h *= del;
    if (Math.Abs(del - 1.0) < 3.0E-7)
      return Math.Exp(-x + a * Math.Log(x) - gln) * h;  // Close enough?
  }
  throw new Exception("Unable to compute GammaUpperCont " +
    "to desired accuracy");
}
```

Figure 8 **The Runs Test**

```
static double RunsTest(BitArray bitArray)
{
  double numOnes = 0.0;
  for (int i = 0; i < bitArray.Length; ++i)
    if (bitArray[i] == true)
      ++numOnes;

  double prop = (numOnes * 1.0) / bitArray.Length;

  // Double tau = 2.0 / Math.Sqrt(bitArray.Length * 1.0);
  // If (Math.Abs(prop - 0.5) >= tau)
  // Return 0.0; // Not-random short-circuit

  int runs = 1;
  for (int i = 0; i < bitArray.Length - 1; ++i)
    if (bitArray[i] != bitArray[i + 1])
      ++runs;

  double num = Math.Abs(
    runs - (2 * bitArray.Length * prop * (1 - prop)));
  double denom = 2 * Math.Sqrt(2.0 * bitArray.Length) *
    prop * (1 - prop);
  double pValue = ErrorFunctionComplement(num / denom);
  return pValue;
}
```

Create Your Own .NET Tool

With the sample code and explanation in this article, you should be able to create a standalone .NET tool to analyze patterns for randomness or integrate pattern-analysis code directly into a .NET software system, using the three tests in the demo program. Additionally, you should be able to implement any or all of the other 12 tests described in the NIST Test Suite document.

Combining Strengths

In technical articles, I generally try to stick to objective information and avoid spouting subjective opinions. But I'll step up on my soapbox this time. I've worked in both government public-sector software development environments (primarily projects for the U.S. Navy while living in Hawaii), and private-sector environments (primarily for Microsoft).

It's easy for private-sector employees to take pot shots at the public sector by calling out things like massive bureaucracy, ponderous documentation and glacial development progress. And it's easy for public-sector employees to take pot shots at the private sector by citing such factors as shoddy quality and wild-west lifecycle processes.

The tests of randomness developed by the National Institute of Standards and Technology provide a good example of how combining the relative strengths of the private and public sectors can lead to better software. I wish that every documentation resource I use was as thorough and clear as NIST documentation. And developing fairly complex mathematical software using C# is dramatically more efficient, less error-prone and quicker than using older technologies. ■

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

Thanks to the following technical expert for reviewing this article: *Dan Liebling (Microsoft Research)*

LIVE EVENT

AMPLIFY YOUR KNOWLEDGE

Live! 360 is back to turn your conference experience up to 11. Spanning 5 days at the Royal Pacific Resort in sunny Orlando, Live! 360 gives you the ultimate IT and Developer line-up, offering hundreds of sessions on the most relevant topics affecting your business today.

SUPPORTED BY

Microsoft | SharePoint | Microsoft SQL Server 2012 | Visual Studio | msdn magazine | Visual Studio MAGAZINE

PRODUCED BY

1105 MEDIA

- 5 Days.
- 4 Events.
- 20 Tracks.
- 175+ Sessions.
- The Ultimate IT and Developer Line-up.

BUY 1 EVENT, GET 3 FREE!

EXCLUSIVE: $800 OFF
Government Employee Discount

To take advantage, call 541.346.3537 TODAY.

Registrant must have a .gov or .mil email address to qualify.

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

ORLANDO | NOVEMBER 18-22, 2013

Royal Pacific Resort at Universal Orlando

LIVE! 360

IT EVENTS WITH PERSPECTIVE

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

Calling all .NET Developers! We've got your ticket to Code for the final stop on the Visual Studio Live! 2013 Tour. **vslive.com/orlando**

SharePoint LIVE!

TRAINING FOR COLLABORATION

Collaborate and Listen. SharePoint Live! returns to rock its newest release, SharePoint 2013. **splive360.com**

SQL Server LIVE!

TRAINING FOR DBAs AND IT PROS

Bringing SQL Server to Your World. Fine tune your skills to solve your biggest data challenges at SQL Server Live!. **sqllive360.com**

Modern Apps LIVE!

MODERN APPS FROM START TO FINISH

The Future of Software Development is Back, Modern Apps Live! will break down the latest techniques in low cost, high value application development. **modernappslive.com**

CONNECT WITH LIVE!360

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

live360events.com

Geo-Visualization of Government Data Sources

Malcolm A. Hyson

When you work with multiple government clients, you'll likely encounter common issues spread across the different organizations. I routinely see multiple efforts revolving around search, content targeting and data visualization. In this article, I'll cover a way to handle geographic data queries using a cloud-hosted API. Then I'll show how you can visualize the results using the Bing Maps SDK for Windows Store apps.

Solution Components

Last year I got a suggestion from a colleague to produce a mobile app that would provide publicly available data based on a user's current location. We went back and forth on ways our end user would interact with the content and decided that a geo-referenced query capability would be the best way to go. With a little research, I found that similar features existed in a few apps, but I was surprised at the lack of apps focused on delivering publicly accessible, geo-coded government information.

Data Sources I researched what types of data existed out there in the wild and quickly found some promising sources, with some surprises regarding the volume of data and its quality. Even after I found what I considered to be a glut of content, I continually got new suggestions from my counterparts working on other government agency engagements. As you'd expect, when pulling content from multiple diverse locations, the one thing you won't get is a standard format. Also, I found that many of the sites varied in relation to the variety and fidelity of content, but most of it came in a few well-known document types such as CSV, GeoRSS, Keyhole Markup Language (KML) and shapefiles.

Once I had a few data sources in mind, I had to determine what components I needed to connect my end users with a reliable hosted service. Here are the tools I selected:

- **Bing Maps SDK for Windows Store apps:** For client-side rendering of maps and geo-referencing search results.
- **Team Foundation Service:** Continuous integration with Windows Azure Web Sites and source control management (SCM).
- **Windows Azure:** Windows Azure SQL Database (formerly SQL Azure) for the data storage and geo-query capabilities; Windows Azure Web Sites to host the API.
- **Windows Store SDK:** Application deployment and framework.
- **Visual Studio 2012 (Update 3)**

Choosing a Data Store and API Although I could write my own app to query and return results using the various aforementioned formats from different sources, there's no reason to reinvent the wheel. The SQL Server platform supports the ability to use spatial queries, so I decided to use it to take care of any data normalization and content query issues. Leveraging this structure allows me to quickly convert any result set into an XML or JSON data feed or any other format that's easily rendered by the geo-display engine of my choice.

Code download available at archive.msdn.microsoft.com/mag201310gGeo.

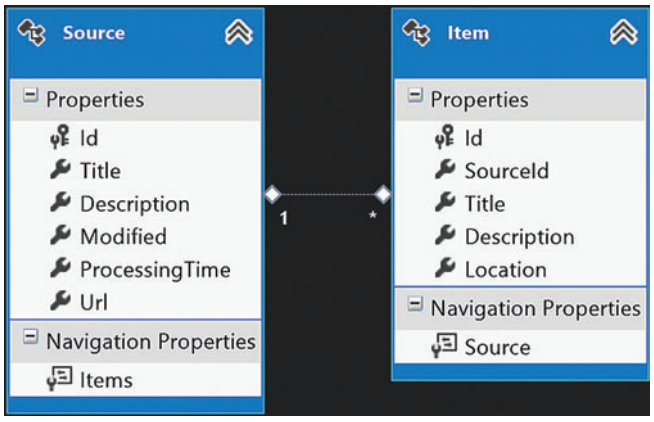


Figure 1 **My Schema Definition Table Structure in the Visual Studio Entity Model View**

I decided to use an ASP.NET MVC 4 Web app with the Web API template to deliver the results to the end client. ASP.NET MVC 4 includes Entity Framework 5 with spatial query support that provides a reliable way to query content stored in SQL Server based on a user-selected, geographically referenced point.

Next, I needed to choose where to host my app, and I decided to go with the Windows Azure Web tier. Visual Studio contains built-in hooks to publish ASP.NET MVC projects to a Windows Azure Web Site. This allows me to take advantage of the free Web application tier along with the ability to easily scale the app if there's sufficient demand.

The Client App After I defined my API, the next step was to devise a way for end users to consume it. Because this is a geo-referenced app, the Bing Maps service was a natural choice to use to render the results on a map. Then I decided to go with a Windows Store app to take advantage of the client deployment and update models. The app created here could be published to the Windows Store with some additional polish, or placed in an enterprise app store if you wanted to manage the deployment of the code within your organization.

Source Control I won't go into much detail related to the application lifecycle management (ALM) of the service; however, in keeping with the cloud-hosted theme, I chose to manage the solution with Team Foundation Service for a few reasons. First is the obvious need for source control and task management. Then there's the issue of basic infrastructure maintenance cost. Although Visual Studio Team Foundation Server 2012 is a pretty easy server application to set up and maintain within your own infrastructure, like any service it comes with a certain amount of management overhead. A few things to think about are OS cost, network infrastructure, security and scalability. Leveraging my own infrastructure can take care of these issues, but it becomes hard to justify when comparing the capabilities available with Team Foundation Service that come with a negligible amount of effort to set up and utilize the platform. All you have to do is sign up using your Live ID and create a new project, and you're off and running. With that said, I'll use Team Foundation Service unless I find a well-defined argument against it.

Building the App

Provisioning the Windows Azure Web Site As stated earlier, I decided to use a Windows Azure Web Site to host the API tier, leveraging an ASP.NET MVC 4 Web API app in conjunction with the SCM capabilities found in Team Foundation Service. Because the back-end service will be a simple Web-hosted API communicating with a SQL Server database, I can quickly get a Web project up and running using the free Web Site option available from the Windows Azure management console. I like to use this option when doing rapid application development and prototyping because of the low barrier to entry coupled with the ability to leverage a highly scalable

Debrief: Government-Hosted Geo-Tagged Content

Different government organizations have published geo-tagged data in a variety of formats and levels of detail. This article describes how to find and then subsequently normalize that content. It also covers how to publish the solution using the Microsoft Windows Azure hosted service with the visualization handled on a Windows Store app integrated with Bing Maps to provide a highly scalable app at a minimal cost.

IT Brief:

By leveraging Software as a Service (SaaS) services when possible, IT managers can drastically reduce or potentially eliminate the amount of infrastructure required to deliver solutions to their customers. Services such as Windows Azure alleviate the need to focus on keeping servers up and online. By leveraging these tools, organizations may quickly deploy and scale applications in a matter of minutes, using the following:

- Windows Azure Web Site publishing
- Team Foundation Service application lifecycle management (ALM)
- Geo-visualization with Bing Maps
- Leveraging a SaaS model to increase the speed of delivery and reduce infrastructure cost

Dev Brief:

Using Windows Azure SQL Database with geographic fields to normalize content from multiple, publicly available data sources can provide many interesting opportunities for data analysis and visualization. By leveraging Team Foundation Service in conjunction with the continuous integration option when connected to Windows Azure, solution developers can quickly build, test and deploy their apps without ever leaving the Visual Studio IDE. This involves:

- Team Foundation Service ALM
- Bing Maps SDK for Windows Store apps
- Windows Store apps
- Windows Azure SQL Database geographic query capabilities

More Information:

- Team Foundation Service: bit.ly/SlfGJ8
- Bing Maps SDK for Windows Store apps: bit.ly/13svBhM
- Rick Strahl's blog post, "Basic Spatial Data with SQL Server and Entity Framework 5.0": bit.ly/MuaLUS
- "Walkthrough: Publishing an F#/C# MVC Application to Windows Azure": bit.ly/18hQW02

architecture. While I'm using a nonsecured site for this application, you could easily add an SSL certificate to encrypt the client communication with the Web service. To learn more about securing API endpoints, see the Windows Azure documentation, "Configure an SSL Certificate on an HTTPS Endpoint," at bit.ly/ueHwrT. To sum it all up, you can scale the app up as it grows and starts to consume more resources. More important, it's free to start, and I've yet to find a price better than free from a hosted-application perspective.

When you create the Windows Azure Web Site, you'll be presented with a few options. I chose to use the custom Web Site option with a new database. Another detail worthy of note is the "Publish from source control" option in the site-provisioning wizard. The combination of the Windows Azure Web Site in conjunction with the continuous integration capabilities of Team Foundation Service provides the ability to quickly code, test and publish the app with minimal effort.

You have a variety of source control options to pick from when you publish from source control, and I selected the default Team Foundation Service option. The integration of Visual Studio and Team Foundation Service allows me to publish my application using continuous integration, so as I check in code, my app will be automatically deployed to a Windows Azure Web Site as part of the build process. To get more details, see the Windows Azure documentation, "Continuous delivery to Windows Azure by using Team Foundation Service," at bit.ly/NOhWAp. I highly recommend taking a little time to read and understand this tutorial if you plan to take advantage of these features.

Defining the Data Store In an attempt to keep my structure as simple as possible for the first iteration, I decided to keep my schema definition to a minimum. My Source table holds the details related to my different content sources, and my Item table holds the details related to my entity of interest. **Figure 1** shows the table structure in the Visual Studio entity model view.

The key to the app is the Location field of the Item table, which uses the sys.geography type. I'll use this field to handle my coordinates for the items in the Item table. I'll also use it to determine the distance from my query coordinates at run time.

Deploying the Database With the first revision of my app, I plan to keep my database changes to a minimum, so I decided to publish my schema from my database project. Visual Studio 2012 provides all of the tooling needed to publish to Windows Azure SQL Database and manage the changes for the duration of the application lifecycle. I like the SQL Server Database Project in this case because it allows me to manage the database schema just like the rest of my source code.

One nuance to using the Database Project with Windows Azure SQL Database is that you need to let Visual Studio know this is a Windows Azure SQL Database Project—as opposed to a regular SQL Server instance—so it can properly deploy. You do this by opening the properties for the Database Project and selecting Windows Azure SQL Database as the target platform.

At this point, I published my schema to the Windows Azure SQL Database instance via the Visual Studio IDE by right-clicking my Database Project and selecting the Publish menu option. Once I enter the information required to point the publishing wizard to my Windows Azure-hosted database, I like to save my publishing profile for future use. This creates a .settings file with the relevant details to publish to a Windows Azure SQL Database instance in the future. Typically, I like to set the default publishing profile of my Database Projects during the development of the data tier to ensure quick and reliable schema synchronization with the current projects. This is accomplished by right-clicking the .publish.xml file and choosing the Set as Default Profile option. Now when I want to publish, my Windows Azure SQL Database values will be prepopulated.

With the settings in place, pushing the schema over to my Windows Azure SQL Database instance couldn't have been easier. Within a few seconds of selecting publish, I was presented with a message stating the request completed successfully.

Adding New Content For most data-driven applications, securing accurate content can be a daunting effort. Fortunately, there are many government geo-referenced data sources available for public consumption. The volume of information can almost seem overwhelming

Figure 2 **Some Sites That Provide Good Geographically Tagged Content**

Site	URL	Description
Data.gov	data.gov	Data from the executive branch. Part of the current administration's open data policy. Focused on delivering federally approved content with the inclusion of other data sets from state governments and universities.
National Atlas	1.usa.gov/ro1cu	Good for geo-referenced bulk stats such as crime and environmental data over time.
National Atlas (1 Million-Scale Data)	1.usa.gov/YpGUyZ	High-resolution data from National Atlas. Data from this part of the site is subjected to a higher level of scrutiny than the other datasets. Data from this part of the National Atlas is also available in a publicly accessible Web service at 1.usa.gov/vpUfAQ.
U.S. Geological Survey (USGS)	usgs.gov	The USGS provides an excellent resource for naturally occurring events within the United States. Here you'll find information on topics such as climate change, influenza outbreaks, earthquakes, rainfall and other events that occur within the United States.
Recovery.gov	1.usa.gov/17mWgKZ	A source for content related to government funding initiatives. This site contains content that details government spending by region.
U.S. Census Bureau	1.usa.gov/1a0a43G	No list of U.S. government data sources would be complete without including content from the Census Bureau.
Naval Oceanography Portal	1.usa.gov/bkII5	Lots of content related to aquatic conditions. The United States Naval Observatory also provides some high-quality astrological data.
Office of Coast Survey (OCS)	1.usa.gov/17iNiSU	I was extremely surprised by how much content was provided by OCS. If I were to do anything with maritime vessels, I'd probably use this data source.

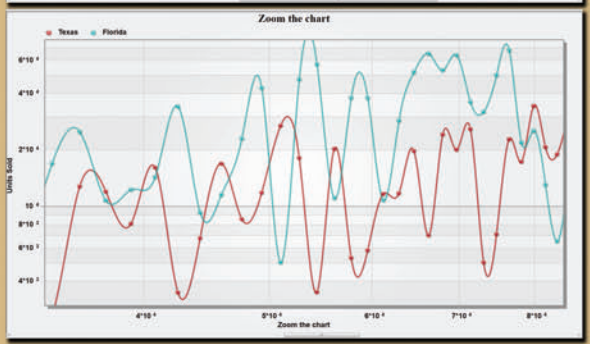
Heighten your Senses.

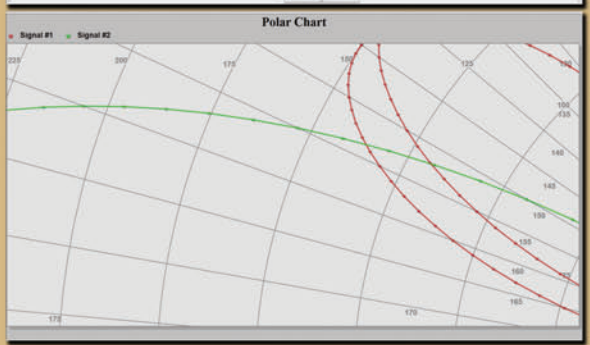
Make no assumptions.

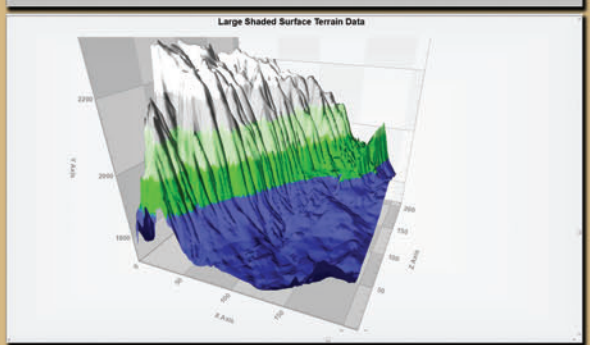
Your projects deserve your best.

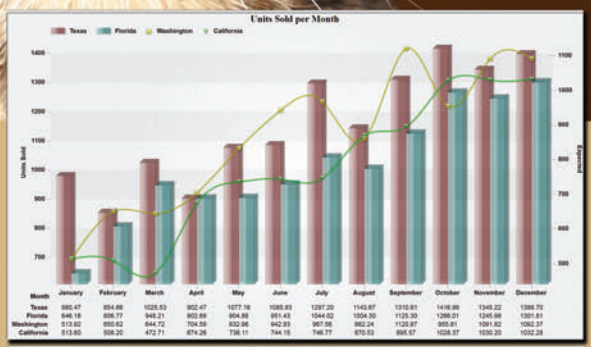
See the 17M no-hassle click-once demo.

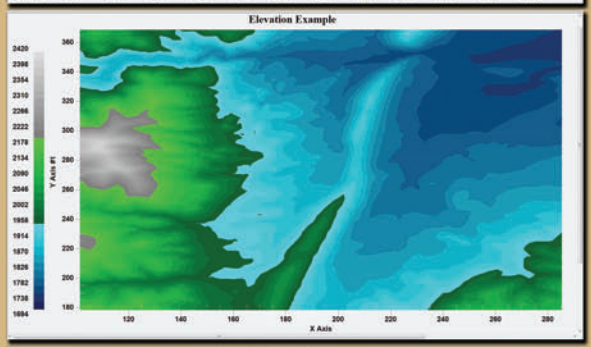
GIGASOFT.COM

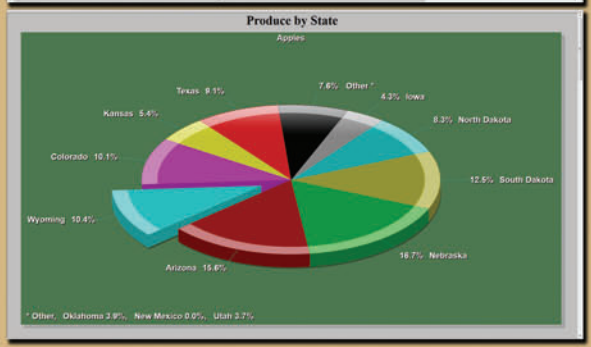












NEW VERSION

ProEssentials v8 now with WPF, Direct2D/3D, and includes WinForms, WebForms, ActiveX, MFC/DLL, and VCL interfaces.

As always, super easy to use and maintain.

With over 15 years history serving the US Dept of Commerce, DOD, NUWC, NOAA, CDC, USGS, Sandia, Lawrence, and Oak Ridge National Lab, USDOE, NIST, DOT, NHTSA, USDA-ARS, FAA, and Social Security Adm.

Gigasoft, Inc. 817 431 8470
email: support@gigasoft.com

Figure 3 **The GetItems Function Returns Items Within a Specified Radius from the Latitude/Longitude Variables**

```
public IEnumerable<SearchResult> GetItems(
  double latitude, double longitude, double radius, string text = "")
{
  DbGeography point = DbGeography.PointFromText(String.Format(
    "POINT({0} {1})", longitude, latitude),
    DbGeography.DefaultCoordinateSystemId);
  double distance = MilesToMeters(radius);
  List<SearchResult> results = new List<SearchResult>();

  db.Items.Where(
    i => (i.Title.Contains(text) || i.Description.Contains(text)
      || String.IsNullOrEmpty(text))
      && i.Location.Distance(point) < distance)
    .ToList()
    .ForEach(r =>
    {
      results.Add(new SearchResult(r));
    }
  );

  return results.AsEnumerable();
}
```

when you start to look into what's out there, but trust me, it's well worth your time to check out these sites and familiarize yourself with their content for future reference. **Figure 2** lists some sites that provide interesting and diverse content.

When I started working with geo-referenced data sources back in the late '90s, the dominant platform in the market was from Esri. That company is still a fairly large player in the geo-visualization arena, and you'll more than likely run into plenty of content delivered using its shapefile format (.shp), which most Esri applications can quickly consume. Because SQL Server doesn't support the native conversion of shapefile files to any type of structure readable by SQL Server, I had to find a way to extract the content from the structure into an easier-to-consume format. As with most interesting problems, there's an interesting solution. Morten Nielsen has provided a free utility called SQL Server 2008 Spatial Tools (bit.ly/3X9IUJ), with which I've had pretty good success. If you don't want to use his code in your environment, Nielsen has provided links to other support tools.

For content, I decided to use the "North American Atlas - Populated Places" (1.usa.gov/13CgufS) download for starters. Using the SQL Server 2008 Spatial Tools application, I uploaded the shapefile file to its own temporary table and then used a select statement to insert it into my defined format, which quickly yielded 1,858 records with which to work. With my database deployed and full of good content, I was ready to create the API.

Building the API

Now that I had my data definition and some content in place, it was time to create the API layer with which my client apps would interact. I decided to go with an ASP.NET MVC 4 Web API project because of its built-in ability to deliver responses in JSON or XML formats. Going this route allowed me to skip all the serialization plumbing I used to have to construct myself to ship my results to the client.

For the Web API, I started by using the ASP.NET MVC 4 Web Application template and selecting the Razor view engine. I chose to leave the default option for "create unit test" selected. While I won't go into detail on unit testing here, it's important to note that as you take your app through a full lifecycle, having the ability to functionally validate your app via the unit-test framework tools provided in Visual Studio shouldn't be overlooked. Once I had the project in place, I needed to add the connectivity to the data store prior to setting up the API commands.

Entity Framework 5 and Web API Entity Framework 5 introduced support for spatial queries, which—in addition to all of the other benefits you gain by leveraging the framework—made it an easy choice to use for integration to my Windows Azure SQL Database to drive the spatial API query support. To do this, I created the model by right-clicking the Models folder, adding a new ADO.NET Entity Data Model (EDM), and following the rest of the framework wizard's steps in order to connect my Web API project with my Windows Azure SQL Database instance.

Thinking through the security of the app, I decided to start off with only providing the ability to query, because this API would be publicly accessible to anonymous users. If I were going to include any PUT or DELETE operations with this release, I'd need to secure those operations to ensure my data could only be modified by authorized users. To provide my query capabilities, I used the standard controller template and overloaded the GetItems function to allow it to handle my geo-referenced query commands.

My API is centered on one operation: to retrieve the results from the database within the specified radius of the selected latitude and longitude, with an optional text parameter that will match the title or description of the items that meet my geographic range restrictions. **Figure 3** shows my GetItems function implemented within my Items class.

The implementation is pretty straightforward, but there are a few areas worth noting:

Distance Unit of Measurement Entity Framework queries using the Distance function will be returned in metric measures. Because I'm writing this app for consumption within the United States, I decided to let the user enter his radius in standard measures instead, so I had to account for the conversion on the server side.

Serializing the Result It's tempting to just want to send back enumerable list entities from the generated model. Although this is possible, I'd also need to deal with the management of the Source-to-Item Entity relationship serialization. Both JSON and XML serializers will attempt to encode the Item and any referenced objects such as Source, which contains a child property of Items—one of which is the Item you're currently serializing. You can see how things would start to get a bit unwieldy. In short, you're throwing the serializer into an infinite reference loop with the default settings when it attempts to turn your class into a JSON result. One way to overcome this is to tell the serializer to ignore the reference loops altogether. Fortunately, Hongye Sun has provided a great blog post on why this issue occurs, along with several methods on how to fix it (see "Loop Reference handling in Web API" at bit.ly/1amm0y2).

Next, I needed to make a few other adjustments to get the XML serialization working. I've seen a few ways to resolve this, and I tend to go with changing the setter on the entity reference model properties to something other than "public." I normally choose "internal," because it allows all generated code to access the setting within my solution, which eliminates the need to change any of the preset code.

SpreadsheetGear

Performance Spreadsheet Components

SpreadsheetGear 2012 Now Available

NEW!

WPF and Silverlight controls, multithreaded recalc, 64 new Excel compatible functions, save to XPS, improved efficiency and performance, Windows 8 support, Windows Server 2012 support, Visual Studio 2012 support and more.

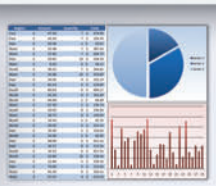
Excel Reporting for ASP.NET, WinForms, WPF and Silverlight

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

Excel Compatible Windows Forms, WPF and Silverlight Controls

Add powerful Excel compatible viewing, editing, formatting, calculating, filtering, charting, printing and more to your Windows Forms, WPF and Silverlight applications with the easy to use WorkbookView controls.

Excel Dashboards, Calculations, Charting and More



You and your users can design dashboards, reports, charts, and models in Excel or the SpreadsheetGear Workbook Designer rather than hard to learn developer tools and you can easily deploy them with one line of code.

Free 30 Day Trial

Download our fully functional 30-Day evaluation and bring Excel Reporting, Excel compatible charting, Excel compatible calculations and much more to your ASP.NET, Windows Forms, WPF, Silverlight and other Microsoft .NET Framework solutions.

www.SpreadsheetGear.com

Visual Studio Microsoft Partner

SpreadsheetGear

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

With all of that in mind, I chose to create a class for returning the results to the client, because there's no reason to return any relational data to the consumer of the API, and it simplifies the serialization of the results. I also decided to pass my Item model to the constructor of my MapSearchResult class to quickly populate the result with the properties from the Item and Source classes that I wanted sent back to the consumer.

Using Spatial Content with Windows Azure SQL Database Anyone who's been around developers long enough has heard the phrase, "It works on my box." For your sake, I'm hoping you hear it used in jest and not as justification to avoid troubleshooting a potential issue. The reason I bring this up is that like any good developer, you'll more than likely create and test locally before throwing your code over the fence to the Windows Azure Web instance, only to see your perfectly developed solution fail miserably when you place it in the cloud and attempt to run a spatial query against your API.

If you configure your Windows Azure instance for tracing, you'll see an error thrown when attempting to load the Microsoft.SqlServer.Types assembly. It turns out that Windows Azure SQL Database is running SQL Server 2012, but it doesn't include all of the out-of-the box binaries, so your app will fail when it attempts to load the aforementioned assembly. Quite a few posts show how to get the required libraries deployed to your Windows Azure instance, but, fortunately, a developer who goes by the handle of "faivre" has put together a nifty NuGet package (bit.ly/141YkKt) that adds the required binaries to your deployment for you.

Deploying the Web API Project It goes without saying that I'd run a few simple tests locally prior to deploying my API project over to Windows Azure. Once I tested my code and basked in the glory of my single-function API, it was time to release it to the world. I personally like to configure the deployment of the local app using a publishing profile associated with the project prior to testing the continuous integration build; however, there are other options. See the MSDN Library article, "Walkthrough: Publishing an F#/C# MVC Application to Windows Azure" (bit.ly/18hQWO2) for more in-depth information. I typically like to download my publishing profile using the Download Publishing Profile menu item on the Web portal and store it with the solution. Once it's downloaded, I select Publish from the project context menu to bring up the publishing wizard. When the publishing wizard opens, you can import your downloaded profile by selecting the Import button.

Getting published to Windows Azure once is great, but because I want this to be a maintainable app across a team of developers, having the developers publish to Windows Azure from their local workstations at will probably isn't the best choice for reliability. I mentioned earlier that I checked the continuous integration option when I created the Windows Azure Web Site. This feature will automatically deploy my app to my Windows Azure Web Site after a successful build. This isn't something to gloss over from the perspective of managing your app in relation to its full lifecycle. I'm really only scratching the surface of what this type of capability allows you to do from a management and deployment perspective (see the aforementioned Windows Azure documentation for more details).

With the tremendous amount of tooling provided by Windows Azure and Visual Studio, it was easy to overlook the capabilities I already had in place prior to writing the client app. I didn't have to worry about details such as acquiring servers, licensing software, getting public IP addresses, installing apps and managing source control. With a minimal amount of code, I could run geo-referenced queries combined with the ability to quickly scale my app up or down as my resource demands changed—not to mention I could easily publish app updates without ever having to leave the comfort of Visual Studio. With the API in place, I needed to visualize these results on a map to see it in action.

Building the Client

For obvious reasons, I wanted to display my geo-referenced query results on a map, and because my API is cloud-hosted, I could choose from a variety of clients to render the results. I decided to go with a Windows Store app, which allowed me to leverage the Bing Maps SDK for Windows Store apps while giving me all of the screen real estate provided to a Windows Store app. After I registered and

Figure 4 **Populating the SearchResults Collection**

```
private async void LoadGeoSearchApiResults(object p)
{
  try
  {
    // Load the results and populate the collection
    this.SearchResults.Clear();

    this.IsWorking = true;
    this.QueryComplete = false;

    HttpClient client = new HttpClient();
    client.BaseAddress = new Uri(this.BaseAddressUrl);
    // Default: http://govdata.azurewebsites.net/

    client.DefaultRequestHeaders.Accept.Add(
      new MediaTypeWithQualityHeaderValue("text/xml"));

    HttpResponseMessage response = await client.GetAsync(
      String.Format(
        "/api/Items?text={0}&latitude={1}&longitude={2}&radius={3}",
      new object[] { this.Query.Text, this.Query.Latitude,
      this.Query.Longitude, this.Query.Radius }));

    using (Stream stream = await response.Content.ReadAsStreamAsync())
    {
      XDocument doc = XDocument.Load(stream);
      XNamespace defaultNs = doc.Root.GetDefaultNamespace();
      var results = doc.Descendants(defaultNs + "SearchResult");

      foreach (var result in results)
      {
        this.SearchResults.Add(new MapSearchResult()
        {
          Title = result.Descendants(defaultNs + "Title").First().Value,
          Description = result.Descendants(defaultNs +
            "Description").First().Value,
          Latitude = Convert.ToDouble(result.Descendants(
            defaultNs + "Latitude").First().Value),
          Longitude = Convert.ToDouble(result.Descendants(
            defaultNs + "Longitude").First().Value)
        });
      }
    }
  }
  catch (Exception ex)
  {
    throw ex;
  }
  finally
  {
    this.IsWorking = false;
    this.QueryComplete = true;
  }
}
```

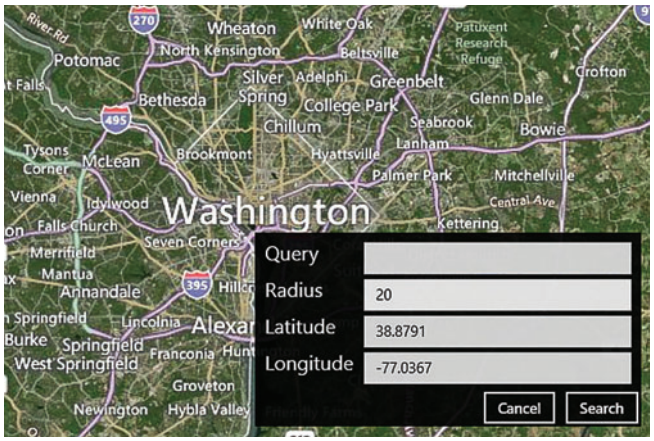


Figure 5 **Displaying the Query Pop-up When the Map Is Tapped**

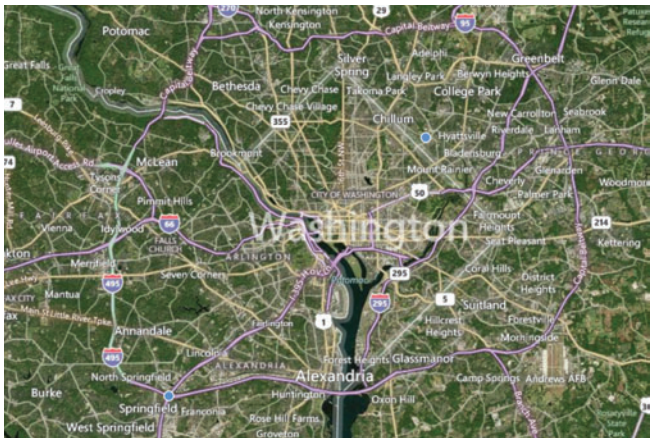


Figure 6 **Results Matching the Query Parameters**

received my Bing Maps key, I was able to create a map with just a few lines of XAML centered on the Washington, D.C., area (latitude 38, longitude -77). I could also start using the end user's current location using the geolocation services with a method provided in the Geolocation sample in the Windows Dev Center (see bit.ly/NnV30u).

Next, I added my search selection pop-up dialog XAML template that would be displayed when a user clicks the map. I created a user control that will show my end users the selected coordinates and then allow them to enter search terms and set the radius to the desired value. So far, so good—but I still needed to connect the map to the API and display the results.

I could quickly jam the code to query the API and lay out the items on the map into the tapped event on the search selection user control, but this could start to become unwieldy as my app grows and I add new capabilities. With this in mind, I opted to go with a Model-View-ViewModel (MVVM)-like design pattern to connect my app to the Web API. By doing this, I could abstract the Web API connection to its own library to potentially reuse it on other clients such as Windows Phone or a desktop client. This also gave me the ability to data bind my XAML definition to the view model, which allows for greater flexibility with the design of my app using the data-bound templates. Even though the MVVM Light Toolkit has become a well-known standard for this type of implementation, I chose not to use it due to the relatively low complexity in my app. If you're going to leverage the MVVM design model in your app, I recommend taking a little time to familiarize yourself with the MVVM Light Toolkit (bit.ly/TkVcM).

The MapSearchViewModel will give my app the ability to query and receive the results from the API. I decided to create a MapQueryResult property to take my user input and transform it into my API query. My SearchResult property is an ObservableCollection of MapSearchResult items. Each item contains the Description, Latitude, Longitude and Title of each search result match. The LoadGeoSearchApiResults async method (shown in **Figure 4**) is where I placed the code to execute the query and populate the result collection. Because this call may take some time, I used an await operator, which allows my UI to stay responsive while waiting for a query to complete and return results. **Figure 4** shows the LoadGeoSearchApiResults function, which handles the call to my Windows Azure-hosted Web API and populates the SearchResults collection.

After creating the MapSearchViewModel, I bound it to the app using a singleton pattern in the codebehind for the MainView. After connecting the view model to the app, I was able to bind it to the query UI and map display with just a few XAML binding statements. Following are two of my XAML snippets for the map control and items binding as a reference.

Here's the Bing Maps XAML:

```
<bm:Map Credentials="{StaticResource ResourceKey=BingMapsKey}"
  x:Name="myMap" MapType="Aerial"  ZoomLevel="5" Tapped="myMap_Tapped">
  <bm:MapItemsControl x:Name="SearchResultItems"
    ItemsSource="{Binding SearchResults}" ItemTemplate=
    "{StaticResource   SearchResultItemTemplate}"/>

  <bm:Map.Center>
    <bm:Location Latitude="33" Longitude="-77" />
  </bm:Map.Center>
</bm:Map>
```

Here's the search result pushpin template:

```
<DataTemplate x:Key="SearchResultItemTemplate">
  <bm:Pushpin PointerPressed="pushpin_Clicked" >
    <bm:MapLayer.Position >
      <bm:Location Latitude="{Binding Latitude}"
        Longitude="{Binding Longitude}">
      </bm:Location>
    </bm:MapLayer.Position>
  </bm:Pushpin>
</DataTemplate>
```

With everything in place, it was time for the moment of truth, and I quickly saw that with a simple tap (see **Figure 5**) of the map my client was correctly rendering my search results from my deployed API (see **Figure 6**).

What I like about these types of geo-referenced map scenarios is the broad applicability across different data types and problem domains. With very little modification, my API serving up geo-referenced queries could be used to augment other apps in a variety of ways, such as notifications based on the user's current location, enhanced imagery displays based on specified coordinates, and object tracking using geo-visualization. I hope this article has shown you a few tips and tricks to aid you along the way to developing and managing your own geo-based solution. ■

Malcolm Hyson *is chief technology officer of Discover Technologies LLC. He's also a newly appointed member of the Visual Studio ALM Rangers team. Reach him at mhyson@discovertechnologies.com.*

Thanks to the following technical experts for reviewing this article: *Soung Bae (Bae LLC) and Westin Kriebel (Microsoft)*

Harness Open Data with CKAN, OData and Windows Azure

Mark Gayler

Open data is becoming increasingly important for governments and public research. Recently some key open data technologies have become available on the Microsoft Windows Azure platform. Comprehensive Knowledge Archive Network (CKAN) is one of the most popular open data platforms in use by governments, universities and enterprises around the globe. Using Windows Azure, open source platforms such as CKAN are able to take advantage of the scalability, flexibility, load-balancing and interoperability features of the cloud. The Open Data Protocol (OData) is an emerging open data standard that enables data to be easily consumed within applications across different platforms, devices and services.

In this article, I'll show you how to deploy a CKAN open data Web site using the virtual machine (VM) capabilities of Windows Azure and VM Depot; how you can access open data published on Windows Azure using the Windows Azure Marketplace; and, finally, how that data can be reused in your applications using OData.

Deploy CKAN in the Cloud

CKAN (ckan.org) is an open source, Web-based data management system that makes data accessible by providing tools to streamline publishing, sharing, finding and using data. CKAN is aimed at data publishers (such as national and regional governments, universities, companies and organizations) that want to make their data open and available. The CKAN project is maintained by the Open Knowledge Foundation (OKFN), and its implementations are commonly used as a public platform for various government data catalogs, including data.gov.uk in the United Kingdom and data.gov in the United States. For example, the U.K. Meteorological Office (Met Office) is using the capabilities of CKAN and Windows Azure to host its weather data archives and make the data broadly available via data.gov.uk/metoffice-data-archive.

Developers who wish to deploy a CKAN installation in the cloud can now do so quickly and easily using the CKAN images available on VM Depot (bit.ly/16PLKxt). VM Depot is a community-driven catalog of preconfigured OSes, applications and development stacks that can be quickly deployed on Windows Azure. VM Depot enables you to find your favorite software and deploy it in minutes, or you can join the community, build a VM image and share it with others.

VM Depot has been developed by Microsoft Open Technologies Inc., a subsidiary of Microsoft. The VM images on the VM Depot site are provided and licensed to you by community members. You can learn more about publishing and deploying VM images in the VM Depot help materials.

The CKAN images on VM Depot are the result of a partnership between Microsoft UK, Microsoft Open Tech and OKFN, which has provided CKAN version 2.1 as a pair of Ubuntu VMs running on Windows Azure. The first image provides the database back end, and the second provides a Web front end. There are two methods of installing CKAN on Windows Azure using these images, as I'll explain in the next section.

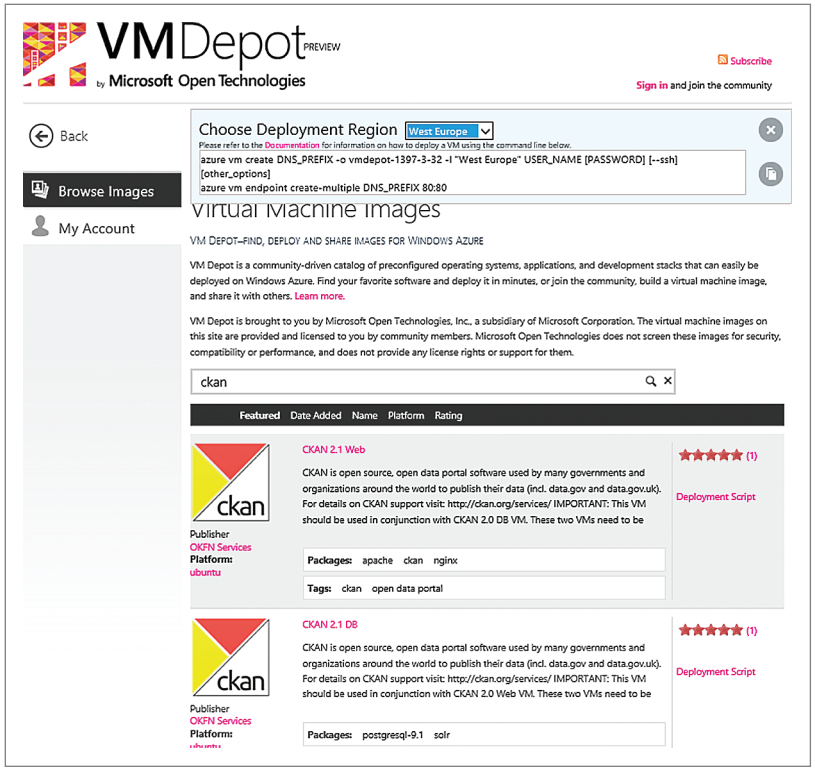


Figure 1 **The VM Depot Interface Showing a CKAN Virtual Machine Listing**

Now I'll walk you through a step-by-step guide to deploying CKAN on Windows Azure. The CKAN project also maintains instructions in the CKAN Wiki on GitHub (bit.ly/16kCPWp).

How to Install CKAN 2.0 on Windows Azure from VM Depot

You can install CKAN on Windows Azure using the images available on VM Depot in two different ways: through the management interface and via the command-line tool. For either method you'll need a Windows Azure subscription; if you don't already have one, you can sign up for a free trial subscription at bit.ly/17pzkur. I'll discuss each method.

Deploying CKAN Using the Windows Azure Command-Line Tool You can find details on how to set CKAN up via the command-line tool in the VM Depot documentation. Once you've installed the tools and configured your Windows Azure credentials, you can publish a CKAN VM. Search for CKAN in VM Depot by typing "ckan" into the search box on the VM Depot site. In your browser, as shown in **Figure 1**, you'll see the two CKAN images. I'll first work with the CKAN Web VM item. To the right of the description you'll see a Deployment Script link. Click the Deployment Script link, click Agree and then select the region to which you want CKAN deployed. A script will appear in the text box. Copy this script and then replace the capitalized text with your specific deployment information. Items in square brackets are optional and—unless you know what you're doing—can be safely ignored for now. Your final script will look something like this:

```
azure vm create ckantest -o vmdepot-1398-1-32 -l
  "West Europe" user pass
azure vm endpoint create-multiple ckantest 80:80
```

The first line of this script creates a new VM with a DNS name that will be available as a VM at DNS_PREFIX.cloudapp.net, for example: "ckantest.cloudapp.net," Linux username "user" and password "pass." The second part opens up port 80 to the outside world. Run both lines of this script.

Now click on the Deployment Script link for the CKAN DB VM item. As you did before, select your region, copy the script and add the same DNS_PREFIX information. Also, it's important to add a -c flag to the command to connect this VM to the previous one. For example:

```
azure vm create ckantest -o vmdepot-1397-2-32 -l "West Europe" user pass -c
```

Debrief: Windows Azure and Open Data

Open data is becoming increasingly important for governments and public research. Using Windows Azure, open data platforms such as Comprehensive Knowledge Archive Network (CKAN) are able to take advantage of the scalability, flexibility, load-balancing and interoperability features of the cloud. The Open Data Protocol (OData) is an emerging open data standard that enables data to be easily consumed within applications across different platforms, devices and services.

IT Brief:

Publishing government data is easy and secure using the cloud.

- The cloud can provide open access to data at low cost
- Open data on the cloud platform is easily accessible by citizens, developers and commercial partners
- Open data in the cloud minimizes impact on local infrastructure
- Open data in the cloud encourages developers to reuse data for development of third-party applications and services

Dev Brief:

Accessing open data in the cloud provides a rich source of information and content for new innovative applications.

- Government open data can be reused in new citizen-centric applications
- Government-published information often includes rich data assets such as transport, geospatial and sensor data
- Use of the cloud and OData enables app developers to easily consume open data in applications running across different devices and platforms, including mobile, tablet and PC

More Information:

- VM Depot: bit.ly/16PLKxt
- CKAN Wiki on GitHub: bit.ly/16kCPWp
- The Open Data Protocol (OData) Specification: odata.org
- Windows Azure Marketplace: datamarket.azure.com
- Newly Available Government Datasets: bit.ly/15bAPvq

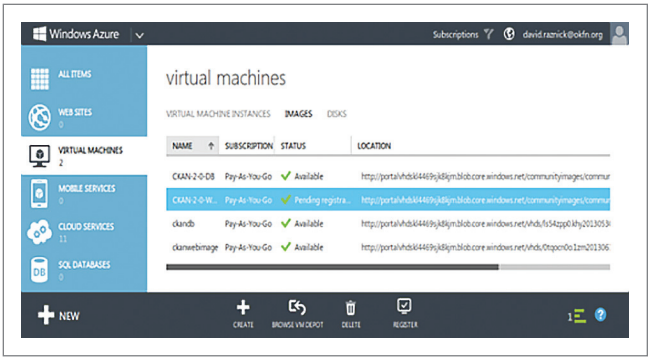


Figure 2 **Registering the CKAN Virtual Machines**

After running this script, CKAN should be available as a VM at DNS_PREFIX.cloudapp.net (for example, ckantest.cloudapp.net).

Deploying CKAN Using the Management Portal Interface

Log in to the Windows Azure Management Portal and click on the Virtual Machines section, then click on the Images tab. Click on Browse VM Depot at the bottom and then find CKAN Web image in the library. Now follow the instructions to install it in the region of your choice. It will take some time to copy this image into your subscription storage account; once you've completed this step, you can deploy as many images as you like. While waiting, you should click on Browse VM Depot again but this time select CKAN DB. Make sure you select the same region for both of them when asked. Once both images have copied to your account, you must register them by selecting each and clicking on the register button at the bottom of the virtual machines page on the portal (see **Figure 2**).

Next, click NEW in the bottom-left corner and select "Virtual Machine from the Gallery." Find the CKAN DB in the image list and select it. Follow the wizard, calling the instance whatever name you like (making sure you remember what you named the VM, for example, "ckanweb"). Provide the DNS prefix so that the machine's DNS name will be useful. For example, if you call it "ckantest," the address will end up being ckantest.cloudapp.net. You can call the user whatever name you like and supply a password or a public Secure Shell (SSH) key. Once the VM has been provisioned, follow the previous steps again for the CKAN Web image. Note that on this occasion, when the wizard asks if you want to create a standalone VM or connect to an existing VM (see **Figure 3**), you should choose to connect to the CKAN DB VM you created previously. Remember, the previous VM must finish provisioning before you can connect another image to it.

Once the wizard is complete and the VMs are provisioned—that is, they have a status of "running"—you need to click on the Web VM to open its details page. Next, click on the Endpoints tab so you can open the necessary port to allow users to access the Web interface. Call the endpoint "web" and put 80 in both the public and private ports fields (see **Figure 4**).

Once this task has finished, your site should be accessible via your chosen domain name (ckantest.cloudapp.net, for example).

CKAN is a useful platform for publishing government data, but you'll often want to analyze the data contained within it. There are a variety of analysis and business intelligence (BI) tools available for this purpose, such as Microsoft Excel. In order to enable such tools to work with a variety of data sources, Microsoft and industry partners have been collaborating on the development of the OData specification (odata.org), which works across a variety of devices and services. In the following section, I'll explore how software developers and data scientists can use OData to access open data published in the cloud.

Windows Azure and OData Overview

Data that's hosted on Windows Azure can also be published using the Windows Azure Marketplace (datamarket.azure.com), which utilizes OData. For example, because the Windows Azure Marketplace provides support for OData, users and developers can easily analyze the aforementioned U.K. national weather data using an OData feed that allows the use of tools such as Power Pivot within Excel. This provides unprecedented access to extensive meteorological data including hourly, daily and five-day forecasts. Based on collaboration between Citrix Systems Inc., IBM Corp., Microsoft, Progress Software Corp., SAP AG, WSO2 Inc. and others,

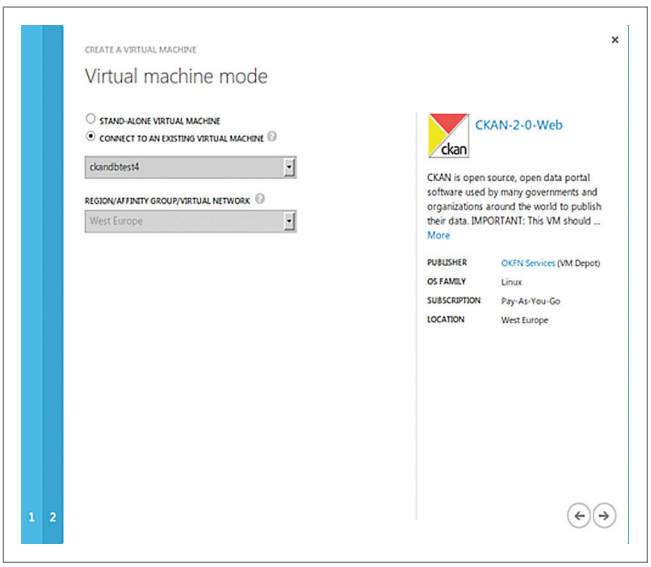


Figure 3 **Connecting to the Existing CKAN Virtual Machine**

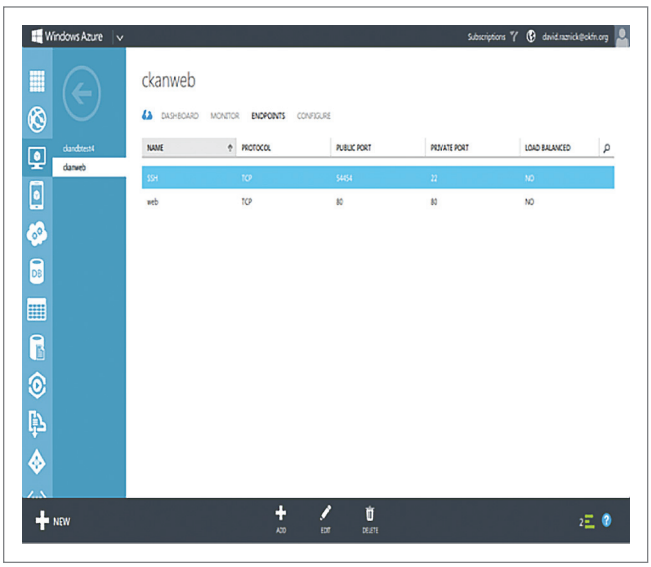


Figure 4 **Opening the Web Endpoints**

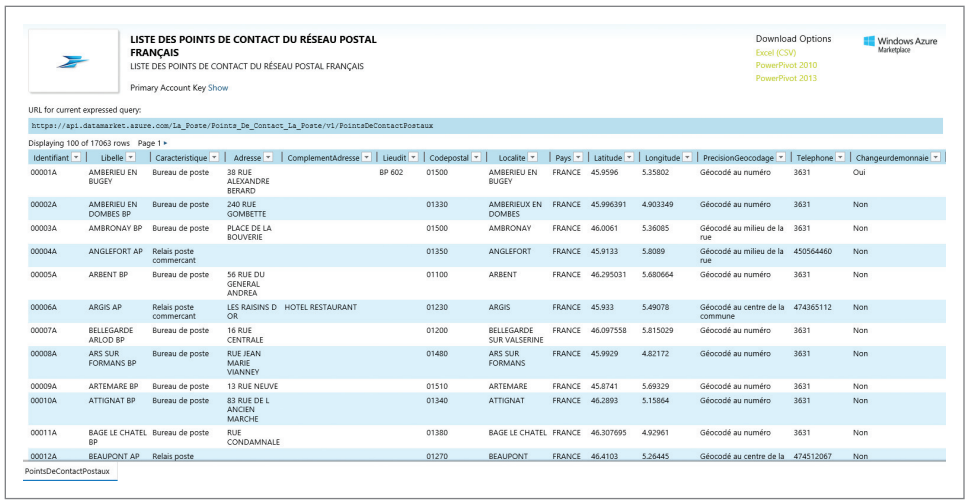


Figure 5 **French La Poste Open Data PointsDeContactPostaux Table Displayed in the Windows Azure Marketplace**

Microsoft Open Technologies recently announced that the OASIS OData Technical Committee initiated a public review of OData 4.0, which is expected to become an OASIS standard this year.

Now I'll show how you can import open data hosted in Windows Azure Marketplace into Excel 2013 via the Power Pivot add-in and OData.

Import Open Data from Windows Azure into Excel Using OData

Open data that's hosted on Windows Azure can easily be imported into an application or service using OData. In this example, I'll show how data can be imported into Excel 2013 using OData and the Power Pivot add-in. You can use this add-in to perform powerful data analysis in Excel 2013. The add-in is available in Microsoft Office Professional Plus and Office 365 Professional Plus editions. It's built in to Excel 2013 but isn't enabled by default. I'll explain how you enable Power Pivot before you use it for the first time. In previous versions of Excel, Power Pivot had to be downloaded as a separate component. In Excel 2013, Power Pivot can be enabled as an add-in using the File | Add-Ins menu. For more information, see the Power Pivot help documentation at bit.ly/17NCVRO.

Accessing Open Data in the Windows Azure Marketplace I'll now go through the steps to get a different government open dataset into Excel using the latest version of Windows Azure Marketplace and Excel 2013. In July, the Data-Market team published a blog post describing some new government datasets available (bit.ly/15bAPvq), including one from the French Postal Office, La Poste. This open dataset, a free subscription, is available in the Windows Azure Marketplace (be aware that some datasets in the marketplace might have specific authentication and login requirements, depending on the nature of the subscription). The Windows Azure Marketplace provides a useful overview page for the data, including your subscription status and more information. When you click on Explore This Dataset, you'll be given the Service Explorer view of the dataset with the default table selected: PointsDeContactPostaux (see **Figure 5**).

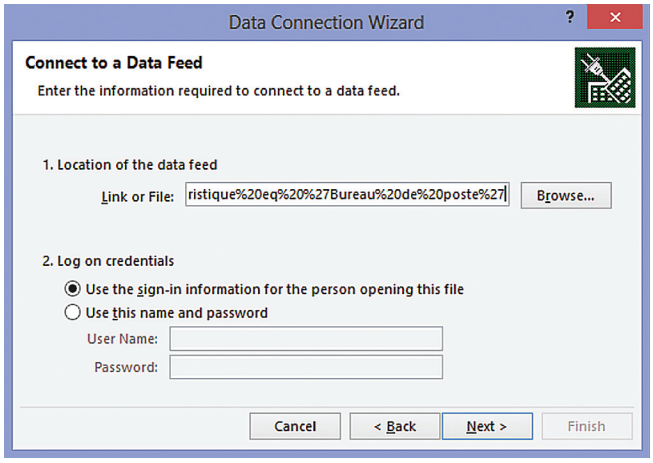


Figure 6 **Importing Open Data from the Windows Azure Marketplace Using an OData Data Feed in Excel 2013**

> Open data that's hosted on Windows Azure can easily be imported into an application or service using OData.

Importing Open Data from Windows Azure into Excel 2013 This data can be imported into Excel 2013 in several ways. For example, you can use the Data Connection Wizard from the Data menu and then select From Other Sources and then From Windows Azure Marketplace. You can also use the Data Connection Wizard to import open data from the Windows Azure Marketplace using OData. From the Data menu, select From Other Sources and then

CKAN and Windows Azure in Yokohama City

There are several examples of governments around the world publishing their data using Comprehensive Knowledge Archive Network (CKAN) and Windows Azure. Yokohama, one of the largest cities in Japan, recently published its open data catalog using CKAN running on Windows Azure. This is the first implementation in Japan of CKAN and Windows Azure for a local open data catalogue. You can see it at data.yokohamaopendata.jp.

From OData Data Feed. The URL required for the OData Data Feed can be found in the Windows Azure Marketplace dataset, "URL for current expressed query" (see **Figure 5**). You can also use the Service Explorer to filter data prior to importing it. In this example, select the dropdown by column Caracteristique and you'll see a dialog box enabling you to filter the column. For this example, I only want Post Office data (rather than retail postal outlets) so underneath "is equal to" enter "Bureau de poste" and then select Filter. Note how the OData query changes with the result. In this example, the URL becomes: https://api.datamarket.azure.com/La_Poste/Points_De_Contact_La_Poste/v1/PointsDeContactPostaux-?$filter=Caracteristique%20eq%20%27Bureau%20de%20poste%27.

Enter that URL in the Data Connection Wizard box, and you'll connect to the data feed in the Windows Azure Marketplace (see **Figure 6**).

However, a more direct and navigable way to get Windows Azure Marketplace open data into Excel 2013 is to use the Power Pivot feature. This gives you more data table manipulation options during the import. Once you've enabled the Power Pivot add-in in Excel 2013 (as described earlier), you'll see the Power Pivot option appear on the top menu. Click on Power Pivot and the Power Pivot menu will drop down. Click on Manage and the Power Pivot dialog box will open. Select Get External Data, then From Data Service, then From Windows Azure Marketplace. At this point, the Table Import Wizard dialog will populate with datasets directly from Windows Azure Marketplace and you can search, filter and select a dataset based on your criteria. You can see government open datasets by clicking on the Government category at the left and then searching for "open data." However, in this case, you'll search specifically for the French Postal Office data. In the Search the Marketplace box, enter "Poste" and then click the search symbol (see **Figure 7**).

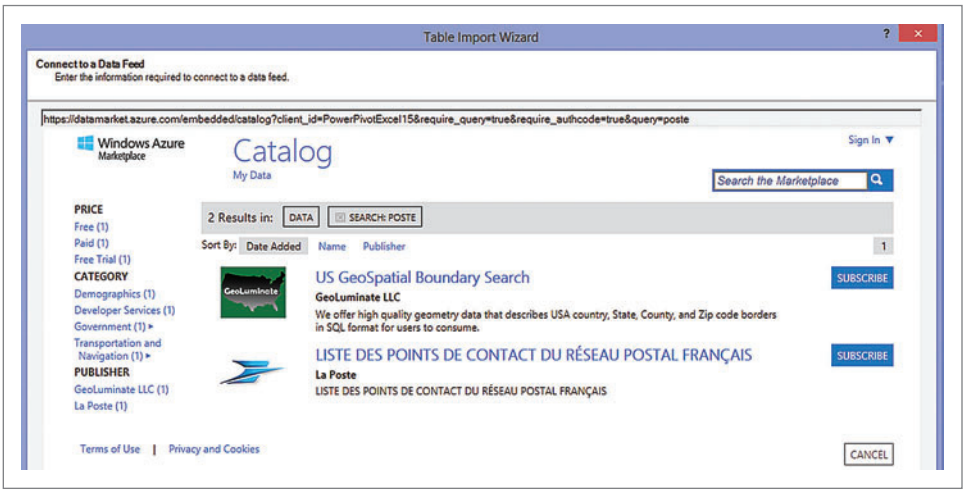


Figure 7 **Selecting Open Datasets from the Windows Azure Marketplace Using Power Pivot in Excel 2013**

Click on the Liste des Points de Contact du Réseau Postal Français link and you'll see the Service Explorer view as displayed in the earlier example. You can still use the Service Explorer to filter data prior to importing it. As before, select the dropdown by column Caracteristique and you'll see a dialog box enabling you to filter the column. In the Filter dialog enter "Bureau de poste" and then select Filter. Click on Select Query and the Table Import Wizard dialog will reappear with a default friendly name for your dataset (change it if you wish). Click Next and the wizard will display the tables you've selected. At this point, you have the option to Preview and Filter if you wish to choose only specific columns to import from your table. Given that you no longer need the Caracteristique column (as you've already filtered on it), you'll use Preview and Filter to omit that

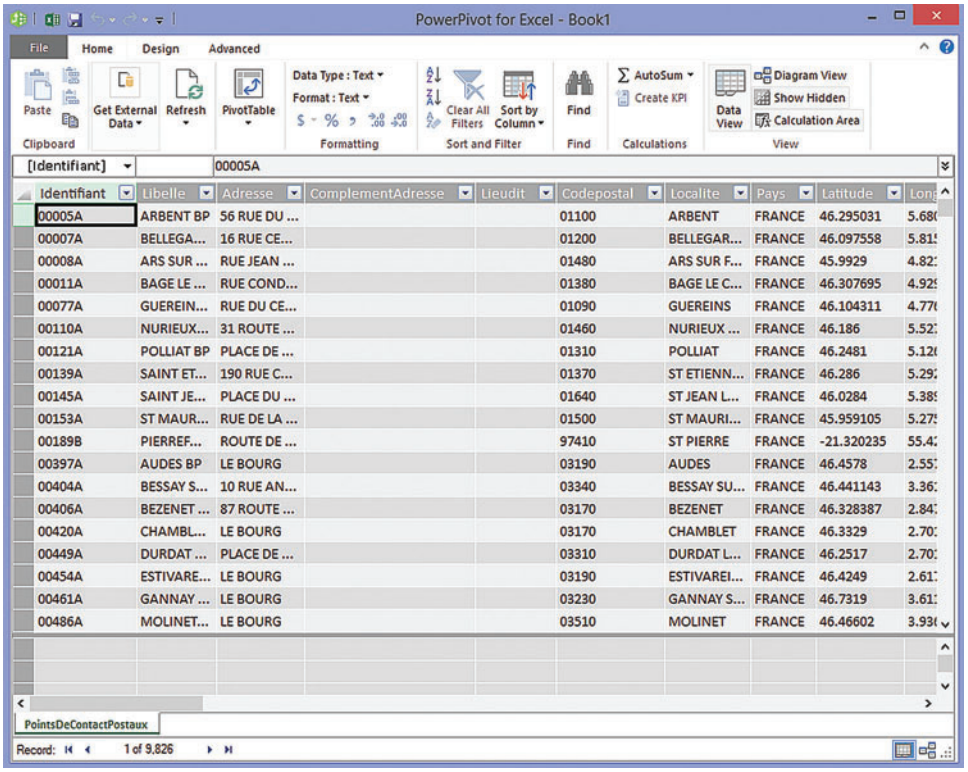


Figure 8 **French Postal Office Open Data Imported from the Windows Azure Marketplace Using Power Pivot in Excel 2013**


Deploy deep
into space with
InstallAware 18
› New InstallAware Direct Deploy Technology. Pushes any EXE without ANY client or server software.
› Complete Unicode support, retaining full backwards compatibility with previous InstallAware projects.
› Bullet-proof, depenency free; run the same setup on Windows XP all the way to Windows Server 2012 R2 x64!
› Supports all Active Directory Domains.
› Competitive upgrade discounts from InstallShield.
www.installaware.com
InstallAware

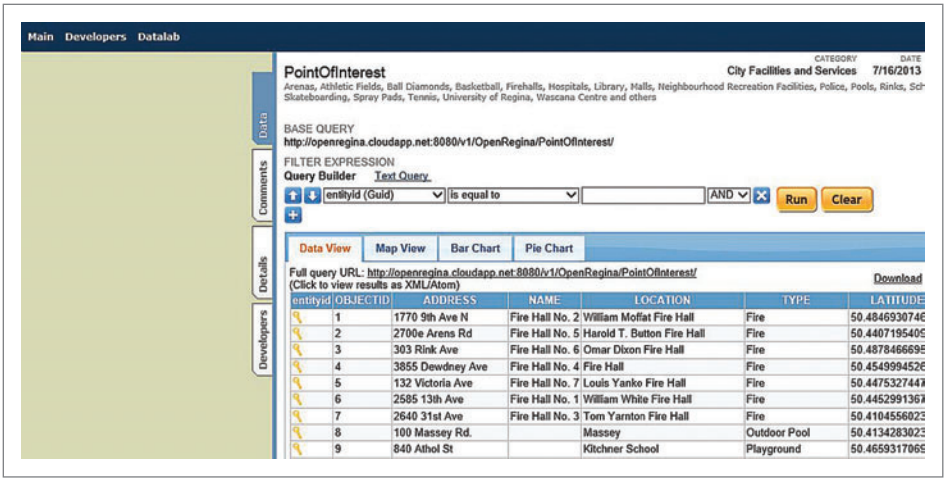


Figure 9 **City of Regina Open Data Catalog Showing the OData Expression Builder**

column during the import. Select Preview and Filter, and when the table columns are displayed, uncheck the Caracteristique column and select OK. Next, click Finish and the filtered PointsDeContact-Postaux table will be imported into your Power Pivot workbook, as shown in **Figure 8** (you'll see a Success! dialog that you need to close).

How to Access Open Data from Catalogs Using OData

So far, I've shown how to set up an open data catalog on Windows Azure using CKAN. I've also looked at how you can import open data from the Windows Azure Marketplace into Excel 2013. Next, I want to explain how you can use the power of OData to incorporate open data into your application code.

To do this, I'll use an external open data catalog built by the city of Regina in Saskatchewan, Canada: openregina.cloudapp.net. The Regina catalog publishes open data using the DataLab API available on GitHub (bit.ly/14M5tAq). DataLab is an open source library that enables governments and citizens to publish open data using Windows Azure, and it provides features targeted specifically at developers who want to incorporate open data within their applications.

In this example, I'll show how DataLab generates application code dynamically to query open data using Windows Azure and OData. This code can be generated in a variety of languages, as you'll see later.

First, go to openregina.cloudapp.net. The first page of the catalog shows the available datasets. You can provide some basic search and filter capabilities here. To start, scroll down and select the PointOfInterest dataset (see **Figure 9**).

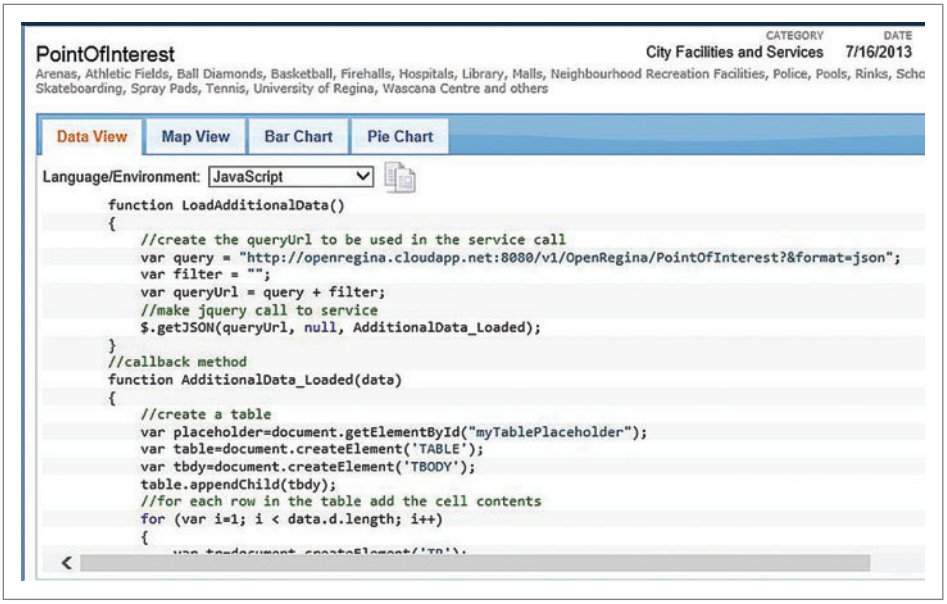


Figure 10 **City of Regina Open Data Catalog Showing a JavaScript Code Sample Based on the OData Expression Builder**

You'll note that in this view, you can see into the dataset (up to 1,000 rows) and you can provide some filtering and querying of the data. Also note the "Full query URL," which will construct an OData feed URL as you query the data. This OData URL can be copied directly into Excel 2013, for example.

This interface also provides dynamic code generation for developers. Click on the Developers tab to see a code interface with a dropdown menu and a code sample provided based on your query selection (and including the corresponding OData syntax). Note that the default developer "Language/Environment" is C#/ASP.NET. Click on the dropdown and you can see the options include Flex, JavaScript, PHP, Ruby and Python. Select JavaScript and you'll see the code sample change dynamically (see **Figure 10**).

The core JavaScript code sample is shown in **Figure 11**.

The code dynamically generated can be used directly in your application to query the open data catalog using OData. Government organizations around the world, including the city of Regina, are publishing open data in the expectation that application developers will build a new generation of innovative applications on the Web, mobile devices and so on.

These initiatives add to the many open data projects based on Windows Azure. Together they facilitate openness and transparency in public data. This work provides an extremely solid foundation upon which open data services can be built. The availability of government- and citizen-published open data represents an opportunity for you to produce innovative new applications and services using a rich data source. When coupled with the work of Microsoft Open Technologies on cross-platform, client-side tooling, you get an environment in which new and exciting opportunities are revealed. ■

Mark Gayler *is a senior technical evangelist at Microsoft Open Technologies, specializing in open data initiatives. Reach him at magayler@microsoft.com.*

Thanks to the following Microsoft technical experts for reviewing this article: *Ross Gardler and Max Uritsky*

Figure 11 **The JavaScript Code Sample Based on the OData Expression Builder**

```
<head>
  <title>Sample Page</title>
  <script src=
    "http://ajax.microsoft.com/ajax/jquery/jquery-1.4.2.min.js"
    type="text/javascript"></script>

  <script type="text/javascript">

    function LoadAdditionalData()
    {
      // Create the query Url to be used in the service call
      var query =
        "http://openregina.cloudapp.net:8080/v1/OpenRegina/" +
        "PointOfInterest?&format=json";
      var filter = "";
      var queryUrl = query + filter;

      // Make jquery call to service
      $.getJSON(queryUrl, null, AdditionalData_Loaded);
    }

    // Callback method
    function AdditionalData_Loaded(data)
    {
      // Create a table
      var placeholder=document.getElementById("myTablePlaceholder");
      var table=document.createElement('TABLE');
      var tbody=document.createElement('TBODY');
      table.appendChild(tbody);

      // For each row in the table add the cell contents
      for (var i=1; i < data.d.length; i++)
      {
        var tr=document.createElement('TR');
        tbody.appendChild(tr);
        tr.appendChild(AddCellContents(data.d, i,'address'));
        tr.appendChild(AddCellContents(data.d, i,'entityid'));
        tr.appendChild(AddCellContents(data.d, i,'gisid'));
        tr.appendChild(AddCellContents(data.d, i,'name'));
        tr.appendChild(AddCellContents(data.d, i,'PartitionKey'));
        tr.appendChild(AddCellContents(data.d, i,'phone'));
        tr.appendChild(AddCellContents(data.d, i,'RowKey'));
        tr.appendChild(AddCellContents(data.d, i,'TimeStamp'));
      }
      // Add the table to page
      placeholder.appendChild(table);
    }

    // Add cell contents to the table
    function AddCellContents(data, cell, id)
    {
      var td=document.createElement('TD');
      var dataCell = data[cell][id];
      td.appendChild(document.createTextNode(dataCell));
      return td;
    }

  </script>
</head>
```

Engage Governments and Communities with Open311

Tim Kulp

Originally, the 311 phone number was designated as a nonemergency call center so that citizens could report issues to local governments or get questions answered. But with the ever-growing presence of smartphones and the ability to access search engines based on GPS location, 311 has evolved primarily for reporting. Open311 (open311.org/learn) is an open standard that's used for tracking civic issues—such as road closures, abandoned vehicles and similar needs.

The Open311 project aims to provide an asynchronous framework through which citizens and local governments can cooperate and converse. This will help citizens notify local governments of incidents and issues needing attention. And Windows Store apps provide a lot of tools with which you can extend Open311 into a platform for civic engagement, collaboration and social action.

Working with Open311

Many cities—including San Francisco, Washington, D.C., and Baltimore—are using robust Open311 implementations. (A complete list can be found on the Open311 site at wiki.open311.org/GeoReport_v2/Servers.) These cities provide APIs developers can use to pump their Open311 implementations full of data from users. This data can then be reviewed, analyzed and assessed by anyone with access to the API (cities often provide open access to their APIs). This data can be used to build and budget community improvement projects, alert authorities to persistent problems (such as the presence of graffiti), or praise city employees.

Code download available at archive.msdn.microsoft.com/mag201310gOpen311.

Building apps around the Open311 API is straightforward. Using REST endpoints, you simply request data via a URL. Open311 has three important components:

1. **Service discovery:** The service discovery document provides data for an Open311 implementation, such as the endpoint URLs for production, testing and staging environments. The service discovery document also identifies the output formats supported by the Open311 implementation. You can see an example at bit.ly/1aWBtEi.
2. **Services:** These are the individual services offered by the Open311 API for a city. Services are organized in categories such as City Employee Praise, Potholes, Graffiti Removal and so on. A complete listing of services is available by providing the service URL without any parameters—for example: http://311.baltimorecity.gov/open311/v2/services.xml. To request the service list in JSON, just change .xml to .json. This updates the output to JSON content instead of XML.
3. **Requests:** These are the issues submitted by users of the system. Requests are associated with a service, which categorizes the request by type. People who submit a request can attach media (such as a photo) as well as commentary. Be mindful that the commentary can be colorful at times, and depending on your content filtering, it can be harmful to your application. Always remember to use input validation and sanitization to mitigate risk to your users and application.

Integrating Open311 into Windows Store Apps

For this sample application, I'll use the Open311 API for my favorite city: Baltimore. I'll demonstrate a small Windows Store app that lets users browse, submit and share requests with others.

To start, I create a new Windows Store app using the Split App template in Visual Studio 2013. This gives you much of the framework for displaying the Open311 data. In the end, you'll have an app resembling **Figures 1** and **2**.

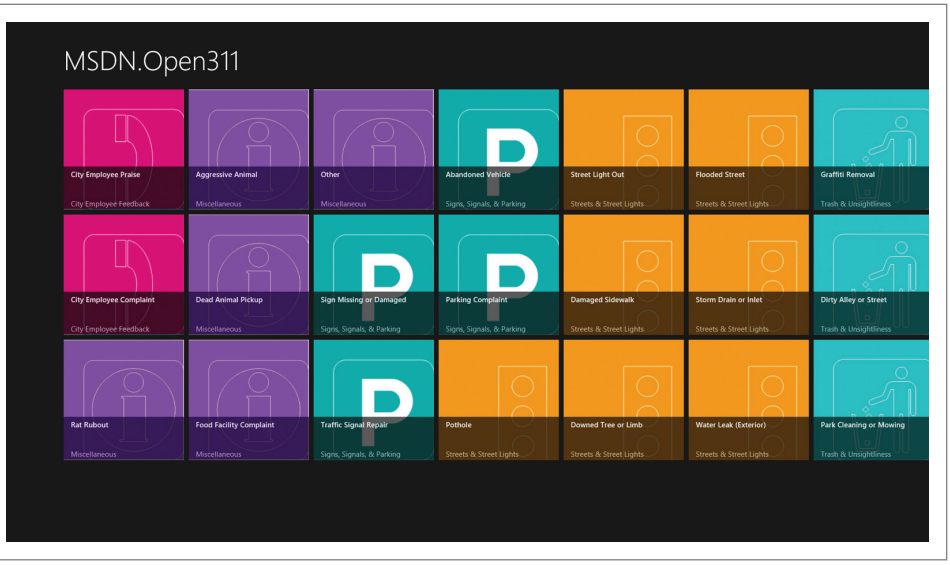


Figure 1 **Open311 Windows Store App Service List Screen**

Using the Split App template gives me a jump on the app's design and code. I get the items page (where the service list appears) and the split page (the request list and details). The split page is where most of the action in the application occurs (such as sharing and creating new requests). I need to add more content, but this is a great start.

The Split App template provides the data.js file as a place to hold and acquire data. I need to modify the data.js file and be sure that the right data is coming into the app. This entails cleaning out the data.js file so all that remains is the anonymous function shell. Then, the following code is added directly under the use strict directive:

```
var serviceList = new WinJS.Binding.List();

var serviceGroups = serviceList.createGrouped(
 function groupKeySelector(item) { return item.group; },
 function groupDataSelector(item) { return item; }
);

var serviceDescription = null;
var requestList = new WinJS.Binding.List();
```

These initial declarations will hold lists for services and requests. Using a WinJS.Binding.List is like using an array but with more features, such as data binding, sorting and grouping. You implement this code to get the service list so users can identify the service they want to see. For this, I'll use the new Windows.Web.Http.HttpClient object to make a call to the Open311 service. The code is shown here:

```
httpClient.getStringAsync(new Windows.Foundation.Uri(
  "http://311test.baltimorecity.gov/open311/v2/services.json"))
  .then(
    function complete(result) {
      var items = JSON.parse(result);
      items.forEach(function (item) {
      item.backgroundImage = "/images/icons/" + item.group + ".png";
      serviceList.push(item);
    });
  }
);
```

The HttpClient object will be my go-to object for connecting to Open311 and getting (or posting) information. The Windows.Web.Http API is new to Windows 8.1, and while not replacing WinJS.xhr, it provides benefits beyond XHR that make it the preferred API for calling data from Open311. To learn more about this library, check out Peter Smith's Build 2013 talk: "Five Great Reasons to Use the New HttpClient API to Connect to Web Services" at bit.ly/13SRQ1N.

Debrief: Engage, Collaborate, Simplify

Open311 provides a communication vehicle for the community to bring social and infrastructure issues to the government. As Gov 2.0 focuses on transparency and trust, Open311 is a perfect model for providing transparency on reported issues and how they're being resolved. Windows 8.1 allows users to have a more connected experience through the Share contract and the engaging UI.

IT Brief:

With today's limited budgets and high expectations, you need to get solutions to market quickly with existing skills. By using JavaScript, you can bring native apps to life on Windows 8 with a modest learning curve, helping you ship apps with your existing team.

- Simplify your app to focus on engagement
- Leverage existing skill sets to keep development costs down
- Use the resources available through MSDN and Visual Studio to speed your time to market

Dev Brief:

Working with Web-based APIs such as Open311 isn't new for most Web developers, but building native apps can be. Using JavaScript and a core set of skills, you can build rich apps that provide engaging experiences. With JavaScript as a first-class code citizen in Windows 8.1, you can develop creative apps with a language you already know.

- Be more creative with apps by focusing on the app
- Leverage Windows 8.1 charms to bring new features to your apps easily
- Work with the Open311 API to dynamically create content in your applications

More Information:

- Open311 Specification: open311.org
- MSDN Dev Center App Inspiration: bit.ly/VwuAzr
- Bing Maps REST API: bit.ly/aHRiWx
- Bing Maps Beta SDK for Windows 8.1 Store apps: bit.ly/1csyG0G

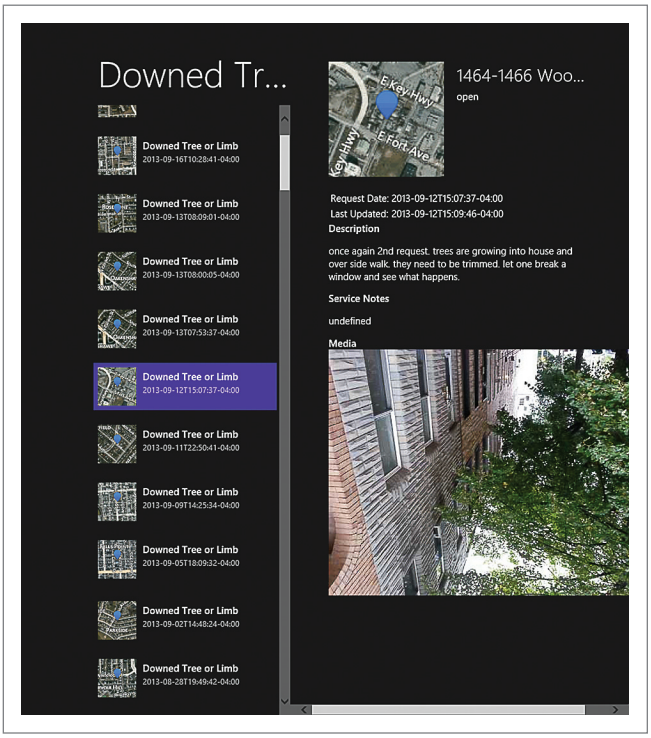


Figure 2 **Open311 Windows Store App Request List/Details Screen**

This object works with promises to perform asynchronous actions, in this case waiting for Open311 to return the list of services. (Programming with JavaScript promises is beyond the scope of this article. Find more information about the promise object and working with it at bit.ly/Lh5Pkp.)

For my purposes, I'll use the then method of the promise object returned by HttpClient.getStringAsync to populate the serviceList. The getStringAsync method returns the response from the service.json page as a string. The string is then passed to the JSON.parse method to convert it into an array of Open311Service objects.

Using JSON.parse is a form of input validation that confirms the content being returned can be converted to a JavaScript object. If the string isn't JSON formatted, the code raises an error. Historically, eval was used to build JSON objects, but this opened the door for script injection. Use JSON.parse to convert text into a JSON object to avoid executing any unknown code.

Any time code interacts with external services, be ready to catch exceptions. When connecting to an external service (in this case, the Open311 server), there's a chance the server will be down or unavailable. Exception handling is omitted here, but can be found in the downloadable sample code.

With the JSON data parsed, the code runs through a foreach loop to add a new property to each service item—in this case, a background image. Adding this property enhances the appearance of the service list so there's a nice grid display with tile images. JavaScript is a prototype-based language, which means that to add a new property, all you need to do is define the property with a value. The service item is then added to the serviceList object—just like adding an item to an array—by using the push method.

With a method to retrieve services, the data.js code now needs a way to retrieve a list of requests associated with a service or services. Requests are the specific incidents reported by the community. Using the Open311 GeoReport standard (wiki.open311.org/GeoReport_v2), there are five parameters a request can support:

- **service_request_id:** The specific ID of the request
- **service_code:** The ID of the service you want to view (this would be used to return all requests for a specific service_code instance)
- **start_date:** Returns all requests since the specified start date
- **end_date:** Returns all requests up to the specified end date
- **status:** Returns requests that are open or closed (the only two permitted values)

To support building a request, the code takes these parameters and formats a request using their values, as shown in **Figure 3**.

Much of this code is the same as the service request in getServices. In the complete function of the then method, the requestList is populated with data from Open311 based on the URL's parameters. For each request item, the Bing Maps API is called for the background image instead of using a standard image from the asset library. This gives the request a point on a map to illustrate the location with more detail.

As you can imagine, the app could do a lot with mapping, and this is where the Bing Maps Beta SDK for Windows 8.1 Store apps would come in. (Learn more about the Bing Maps Beta SDK for Windows 8.1 Store apps at bit.ly/1csyGOG.) To filter which requests are returned, the URL request is built based on the data provided to the function. Some Open311 implementations filter on every parameter provided regardless of whether it has a value. To avoid unintentional filtering here, any parameter that isn't being used is left off.

Figure 3 **getRequests Function**

```
function getRequests(serviceRequestId, status, startDate, endDate, serviceCode) {
  var url = "";
  if (serviceRequestId !== ""){
    url = "http://311.baltimorecity.gov/open311/v2/requests.json?jurisdiction_
    id=baltimorecity.gov&service_request_id=" + serviceRequestId;
  }
  else {
    url = "http://311.baltimorecity.gov/open311/v2/requests.json?jurisdiction_
    id=baltimorecity.gov";
    if (startDate !== "")
      url += "&start_date=" + startDate.toISOString();
    if (endDate !== "")
      url += "&end_date=" + endDate.toISOString();
    if (serviceCode !== "")
      url += "&service_code=" + serviceCode;
    if (status !== "")
      url += "&status=" + status;
  }

return httpClient.getStringAsync(new Windows.Foundation.Uri(url))
 .then(
    function complete(results) {
      var items = JSON.parse(results);
      items.forEach(function (item) {
        if (requestList.indexOf(item) == -1) {
          item.backgroundImage = "http://dev.virtualearth.net/REST/
v1/Imagery/Map/AerialWithLabels/" + item.lat + "," + item.long +
"/15?mapSize=100,100&format=png&pushpin=" + item.lat + "," + item.long +
";65;&key=[BING KEY]";
          requestList.push(item);
        }
      });
  });}
```

With the getServices and getRequests methods defined, the code needs a way to access them. Fortunately, WinJS provides a simple mechanism for creating objects, called WinJS.Namespace.define. An object (in this case "Data") and the members of the object just need to be specified:

```
WinJS.Namespace.define("Data", {
  services: serviceGroups,
  serviceGroups: serviceGroups.groups,
  requests: requestList,
  GetServices: getServices,
  GetRequests: getRequests});
```

Instead of using the serviceList variable, which was populated in the getServices method, the serviceGroups object is used. The serviceGroups object is a "live projection" of the serviceList list. In other words, serviceGroups is a grouped representation of the serviceList. Any changes that occur to the serviceList are propagated to the serviceGroups object so the app can add, remove and update the serviceList without having to reapply the grouping.

Service List Screen (Items Page)

The Split template provides a few screens for the Windows Store app. The first is the Items page, which is used to list the groups of data in the template. This screen will be updated to use the serviceGroups list. The data bindings will also be updated to render the Service object model.

Changes to Items.html The only update needed here is to the data template used to display the service items. In the items.html page (pages/items/items.html), change the binding fields to reflect the Service object model, as shown here:

```
<div class="itemtemplate" data-win-control="WinJS.Binding.Template">
  <div class="item">
    <img class="item-image" src="#" data-win-bind="src: backgroundImage;
      alt: service_name" />
    <div class="item-overlay">
      <h3 class="item-title" data-win-bind="textContent: service_name"></h3>
      <h4 class="item-subtitle win-type-ellipsis"
        data-win-bind="textContent: group"></h4>
    </div>
  </div>
</div>
```

The ListView control uses the template in the main content section to render each service item. For each item, the template displays an image with a text overlay that describes the service. Using the data-win-bind attribute, the code defines which field in the object model to render for each attribute. While the <h3> and <h4> tags are rendering only text, the <img> tag binds the image source and alternate text. You can bind as many attributes as you need by using attribute:field; value.

While on the html page, update the ListView at the bottom of the screen to set data-win-options only to selectionMode: 'none.' The other settings will be handled by the items.js code file.

Changes to Items.js Here, again, use the Split template and just tweak the code. First, the services need to be loaded into the Data object. The getServices method defined in the Data object will be called to populate the serviceList, but only if the serviceList is empty. The serviceList is a WinJS.Binding.List object, which has a lot of the same properties as an array. Check the length of the list to avoid unnecessary loading.

Next, in the ready event handler in items.js, update the ListView object to use the Data.services list for its data binding. Finally, update the _itemInvoked method at the bottom of items.js to use the Data.services object instead of the Data.groups object from the template. Complete code changes can be found in the sample code available for download at archive.msdn.microsoft.com/mag201310gOpen311 and have been omitted here for brevity.

Run the app. The items page lists the Open311 services for Baltimore. The images will be blank because they haven't been defined in the app yet. (These images are included in the sample code.) In the code, the _itemInvoked method is associated with the onItemInvoked event of the ListView.

When the user taps or clicks an item in the ListView, the groupKey is retrieved from the Data.services list and is passed to the

Figure 4 **Create Filtered Projection for Requests**

```
this._items = Data.requests.createFiltered(
  function filterList(item){
    if (item.service_code == serviceItem.service_code)
      return true;
    else
      return false;
  }).createSorted(
  function sorter(item1, item2) {
    var date1 = item1.requested_datetime;
    var date2 = item2.requested_datetime;
    if (date1 == date2)
      return 0;
    else if (date1 < date2)
      return 1;
    else
      return -1;
  });
```

Figure 5 **HTML for Adding a Request**

```
<h2>Adding a request: <span id="serviceName"></span></h2>
  <form id="service-description">
    <p>
      <label for="txtAddress">Address</label>
      <input type="text" id="txtAddress" aria-label="Address for the request"/>
    </p>
    <p>
      <label for="txtDescription">Description</label>
      <textarea id="txtDescription" aria-label="Request description"
        rows="6" cols="45"></textarea>
    </p>
    <div id="custom-content"></div>
    <fieldset>
      <legend>Your contact information (OPTIONAL)</legend>
      <ul>
        <li>
          <label for="txtFN">First Name</label>
          <input type="text" id="txtFN" aria-label="First Name of requestor"/>
        </li>
        <li>
          <label for="txtLN">Last Name</label>
          <input type="text" id="txtLN" aria-label="Last Name of requestor"/>
        </li>
        <li>
          <label for="txtEmail">Email Address</label>
          <input type="email" id="txtEmail"
            aria-label="Email address of requestor"/>
        </li>
        <li>
          <label for="txtPhone">Phone Number</label>
          <input type="tel" id="txtPhone" aria-label="Phone number of requestor"/>
        </li>
      </ul>
    </fieldset>
    <p>
      <input type="button" id="btnSend" aria-label="Save" value="Send Request" />
    </p>
  </form>
```

Figure 6 **Pulling Location Information**

```
var btnSend = document.getElementById("btnSend");
btnSend.addEventListener("click", btnSend_Click, false);
locator = new Windows.Devices.Geolocation.Geolocator();
var locatorPromise = locator.getGeopositionAsync();
locatorPromise.then(
  function complete(position) {
    lat = position.coordinate.latitude;
    long = position.coordinate.longitude;
  },
  function error(ex) {
    // Handle the error
});
```

split.html page through the navigate method's initialState argument. WinJS.Navigation.navigate accepts two arguments: where to go (the string path of the destination page) and any data the new page should receive from the calling page (initialState). In this case, the code passes groupKey to the split.html page, which is the specific Service object activated by the user. The initialState argument of the navigate method lets apps easily pass data from one page to another. Tapping a service at this point will error out because split.js hasn't yet been updated. I'll fix that now.

Requests Page (Split Page)

The items.html page redirects to the split.html page and passes along the Service data the user selects. On this screen, the app displays a list of requests for the selected service as well as details about a specific request. Again, the existing split.html page needs to be modified only to implement the specific functionality.

Developing Split.html This page consists of two parts. The first part is the list of requests based on the selected service from the items.html page. The second part is the detailed view of the specific request selected.

Defining the list requires code similar to what I added to the items.html page. First, reference the code sample and update the template in split.html to reflect the Request data model. Next, replace the pre-generated article section with the code sample's article section to display a specific request item's detail view. This HTML is updated to the Request data model.

Notice the accessible rich Internet application (ARIA) attributes that appear throughout this code. These attributes are used in ARIAs for assistive technologies. ARIA attributes are one of the semantic enhancements in HTML5. Many government organizations have strict accessibility standards for their Web sites and media. You can learn more about the ARIA attributes at bit.ly/19kuuyd.

Figure 7 **Check for Service Item Metadata and Get Service Description**

```
serviceItem = options.serviceItem;

if (serviceItem.metadata) {
  Data.GetServiceDescription(serviceItem.service_code).done(
    function complete(result) {
      if (result.status == 200) {
        var serviceDescription = JSON.parse(result.responseText);
        var dynamicObject = document.getElementById("custom-content");

        serviceDescription.attributes.forEach(function (attribute) {
        if (attribute.variable) {
          // ...
        }
      });
    }
  });
}
```

Finally, AppBar is added to the page. AppBar is where users create a request to send to Open311. A single Add button allows users to add a new request, but other buttons can be added later, such as Refresh to pull updates from the server:

```
<div id="commandsAppBar" data-win-control="WinJS.UI.AppBar" data-win-options="">
  <button data-win-control="WinJS.UI.AppBarCommand"
    data-win-options="{id:'cmdAdd',label:'Add',icon:'add',
    section:'global',tooltip:'Add new
    request'}"></button>
</div>
```

One of the great things about the AppBar class is that it provides a common area for locating app commands. This keeps your app's interface clear of distraction or clutter. For this app, users might want to just browse and not post. To support this experience, place the add functionality in a standard location (the AppBar) away from users who don't want it.

Changes to Split.js To bring these views to life, reconfigure the codebehind page to work with the Data.requests object.

First, at the top of the page, under the declaration of the utils variable, add two new variables the code will reference:

```
var appBar = null;
var serviceItem = null;
```

Now set up the code to bind the ListView to the requests for the selected service. Start by setting the serviceItem to the data that comes from items.html. Notice the ready function has two arguments: element and options. Options is the data sent from the previous page and contains the groupKey object from items.html. In the ready function defined in the split.js file, set the serviceItem to the groupKey:

```
serviceItem = options.groupKey;
```

Throughout the JavaScript code, the serviceItem provides context for which requests the app needs to display to the user. Set the title of the page to reflect the service selected by the user:

```
element.querySelector("header[role=banner] .pagetitle").textContent =
  serviceItem.service_name;
```

Now call the getRequests method of the Data object to pull the requests for the Service object. The following function calls for all open requests for the selected service. Add this call at the top of the ready event handler in split.js:

```
Data.GetRequests("", "open", "", "", serviceItem.service_code);
```

As the user moves between services, the request list grows with data for the various services. Instead of clearing out the requests each time the user comes to the split page, just build a filter to display only the data for the selected service. By using createFiltered and createSorted (see **Figure 4**), another live projection (as with the serviceGroups) is created on top of the WinJS.Binding.List object. As data populates the list, the filtering and sorting projections remain so that they don't need to be reapplied after the data is loaded.

The createFiltered method loops through each item and checks the service_code value of the Request object to determine whether it matches the service_code of the serviceItem. If it does, the Request object is represented in the projection. If not, the projection ignores the entry. The filtered list is then sorted by comparing the requested_datetime of each item to determine placement in the projection, with the most recent date displayed at the top of the list to create a descending order for the requests.

Figure 8 **Specific Data Types for a Variable**

```
var rootElement;
switch (attribute.datatype) {
  case "string":
    // Create a label and input box with type text
    break;
  case "number":
    // Create a label and input box with type number
    break;
  case "datetime":
    // Create a label and input box with type datetime
    break;
  case "text":
    // Create a label and a textarea
    break;
  case "singlevaluelist":
    // Create a label and a select list
    break;
  case "multivaluelist":
    // Create a label and select list for each item
    break;
}
```

The request list and details sections are now provided with data. By using the existing code in the split page and updating just the pieces needed for the data model, you have working code very quickly. As a last step at this point, add the cmdAdd_click callback to the bottom of the split.js page, right above the closure of the anonymous function:

```
function cmdAdd_Click(e) {
  WinJS.Navigation.navigate("/pages/create/create.html",
    { serviceItem: serviceItem });
  }
```

Associate this function with the cmdAdd button's click event in the ready function of the split.js page, and the page is ready to be tested:

```
appBar = document.getElementById("commandsAppBar").winControl;
appBar.getCommandById("cmdAdd").addEventListener("click", cmdAdd_Click, false);
```

Creating Service Requests

Users need to be able to create a request for a service and send it to the Open311 framework. Different service requests have different data points that need to be defined, so the app must dynamically render controls based on the type of service. Using the service definition from Open311, you can determine what data fields need to be provided, whether a field is required and possible values.

First, add a method to the Data object in the data.js file, called getServiceDescription, with the following code:

```
function getServiceDescription(serviceCode) {
  return httpClient.getStringAsync(
    new Windows.Foundation.Uri(
      "http://311test.baltimorecity.gov/open311/v2/services/" +
      serviceCode + ".json?jurisdiction_id=baltimorecity.gov"));    }
```

Here, the HttpClient.getStringAsync method's promise object is returned. You can use this to determine how to render the service description. For the Windows Store app, the code takes the result and renders appropriate controls. To make this function available to other pages, make sure to add it to the WinJS.Namespace.define call at the end of data.js as getServiceDescription.

Figure 9 **Control Code**

```
rootElement = document.createElement("p");
var select = document.createElement("select");
select.id = "select" + attribute.code;
var label = document.createElement("label");
label.textContent = attribute.description;
rootElement.appendChild(label);
attribute.values.forEach(function (value) {
  var option = document.createElement("option");
  option.value = key;
  option.text = value;  select.appendChild(option);
});
select.setAttribute("data-attribute-code", attribute.code);
rootElement.appendChild(select);
```

Create a new folder under pages and name it "create." In that folder, add a Page control named create.html to use for submitting a request to Open311. Requests include some standard information, represented here along with a "custom-controls" div populated from the service definition. Use the HTML in **Figure 5** for the input form.

Now, open create.js, and add the following declarations within the immediately invoked anonymous function, under the use strict directive:

```
var serviceItem = null;
var locator = null;
var lat;
var long;
```

Open311 is an ideal location-awareness technology for tracking specifically where an issue occurs. To support this, the app uses a GeoLocator object to get the latitude and longitude of where the incident is reported. In cases when users don't want to provide their exact location, the address field can be populated manually. Inside the ready function, add the event listener for the btnSend button and connect the locator so that it polls a user's location information. In the example shown in **Figure 6**, the location is pulled in the ready function so this information is gathered behind the scenes while the user fills out the request form.

Using getGeopositionAsync, the code returns a Geoposition object with a user's location. This location is derived from a GPS device or the user's Internet connection. When this function is called, the user is prompted to allow or deny the app's access to location information.

In Open311, the user's location must be specified, but this can be done with latitude and longitude or an address. Anytime your app needs data to which a user could deny access, be sure

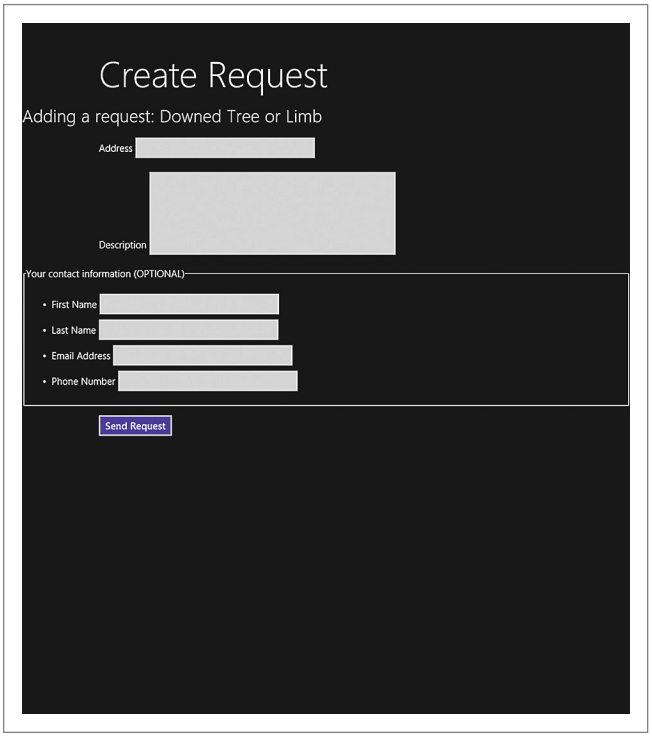


Figure 10 **Open311 Windows Store App Create Request Screen**

Figure 11 **postRequest function**

```
function postRequest(options) {
  var data = {
    api_key: "[YOUR OPEN311 API KEY]",
    service_code: options.serviceCode,
    lat: options.lat,
    long: options.long,
    address_string: options.address_string,
    address_id: options.address_id,
    email: options.email,
    device_id: options.device_id,
    account_id: options.account_id,
    first_name: options.first_name,
    last_name: options.last_name,
    phone: options.phone,
    description: options.description,
    media_url: options.media_url
  }

    options.attributes.forEach(function (attribute) {
      data["attribute[" + attribute.code + "]"] =
      encodeURIComponent(attribute.value);
    });

    var strData = "";
    for (var key in data) {
      if (data.hasOwnProperty(key)) {
        if(data[key] !== "")
          strData += key + "=" + data[key] + "&";
      }
    }

    var httpContent = new Windows.Web.Http.HttpStringContent(strData,
      Windows.Storage.Streams.UnicodeEncoding.utf8,
      "application/x-www-form-urlencoded;
      charset=utf-8");
    return httpClient.postAsync(new
      Windows.Foundation.Uri(
      "http://311test.baltimorecity.gov/open311/v2/requests.json"),
      httpContent).then(
      function complete(result) {
      // Do something with the result
        var something = result;
      },
      function error(ex){
      var e = ex;
    });
  }
```

you build an alternative mechanism that's available should this denial occur. Here, the app has the address field for when location information is denied.

The previous page (split.js) sent create.js the service data. Now you can check the metadata of the service data object to see whether this service item has any additional fields beyond the basic ones. This code is shown in **Figure 7**.

As you can see, a pattern that's used elsewhere in the app is being used. In this case, retrieve the serviceDescription and loop through each of the attribute elements to render the control. The code also checks to see whether an attribute's value is variable—not a fixed value. As an example, for graffiti reports, one of the attributes is "surface," which is a variable that can be set to a variety of values, such as concrete, brick and so on. Attributes also have data types, such as string, number, datetime, text (as in textarea), singlevaluelist (a dropdown list) and multivaluelist (cascading dropdown lists). Replace the ellipsis in **Figure 7** with the code in **Figure 8**.

Creating these controls is simple and is a familiar operation for JavaScript developers. Replace the comment under the string case with the following:

```
rootElement = document.createElement("p");
var textBox = document.createElement("input");
textBox.setAttribute("type", "text");
textBox.setAttribute("data-attribute-code", attribute.code);
textBox.id = "input" + attribute.code;
if (attribute.required)
  textBox.setAttribute("required", "required");
var label = document.createElement("label");
label.textContent = attribute.description;
rootElement.appendChild(label);
rootElement.appendChild(textBox);
```

This is straightforward DOM element creation with values assigned. One difference is that you create your own data attribute to store the attribute object's code value. The code is used later to identify the value of this attribute when the request is submitted to the server. To make the value easier to pull, a standardized value—"data-attribute-code"—is created so it can be queried. While this simple text box is easy to build, the most common attribute data type for Baltimore is singlevaluelist, which renders as a select box. Replace the singlevaluelist comment text with the code in **Figure 9**.

One notable difference in this code and the code for a text box is the use of the values collection of the Open311 attribute object. This is a sequence of key/name pairs that provide a value (key) with text to describe the value (name). In the code in **Figure 9**, the createOption function simply takes the key and name properties of the value object and outputs an option tag with the value set to the key and the text set to the name.

At the end of the switch statement, add the rootElement object to the dynamicObject object to render the controls to the screen:

```
document.getElementById("custom-content").appendChild(rootElement);
```

The result should look something like **Figure 10**.

With this form, users can enter data and click Send Request to invoke the btnSend_click function. To make this function do

Figure 12 **btnSend_Click Signature**

```
function btnSend_Click(e) {
  var description = document.getElementById("txtDescription").innerText;
  var address = document.getElementById("txtAddress").value;

  if ((lat == undefined || long == undefined) && address == "")
    return; // Display an error message to the user

  var options = {};
  options.first_name = document.getElementById("txtFN").value;
  options.last_name = document.getElementById("txtLN").value;
  options.email = document.getElementById("txtEmail").value;
  options.phone = document.getElementById("txtPhone").value;
  options.attributes = new Array();
  var attributeCollection = document.querySelectorAll("[data-attribute-code]");
  for (var i = 0; i < attributeCollection.length; i++) {
    var value = document.getElementById(attributeCollection[i].id).value;
    var code = document.getElementById(attributeCollection[i].id)
      .getAttribute("data-attribute-code");

    options.attributes.push({ code: code, value: value });
  }
  options.address_string = address;
  options.address_id = "";
  options.lat = lat;
  options.long = long;
  options.device_id = "";
  options.account_id = "";
  options.media_url = "";
  options.description = description;
  options.serviceCode = serviceItem.service_code;
  Data.PostRequest(options);
}
```

What About Privacy?

By using the Open311 standard, users submit data to a specific city, just as though they were reporting the data to the city over the phone. What the city does with that data is up to the city, but developers of Windows Store apps need to include a privacy statement that outlines exactly what's happening with the data their apps collect.

Think of building an app for Facebook. The app can collect data, but it will store some of that data on Facebook. After the data goes to Facebook, Facebook can do what it wants to with the data (in conformance with its own usage policies). When you develop and manage apps with Open311, be sure the privacy statement clearly indicates that users are providing data to the specific city for the city's use.

something, you need to add one last function to the Data namespace: PostRequest. To cut down on arguments, use an options object as a single argument with multiple properties. Add the code in **Figure 11** to the data.js file.

First, the code builds a data object that's used to hold the values of the parameters you want to post to Open311. Then you need to include all the attributes in the format "attribute[CODE]=KEY." To do this, the code loops through the attributes collection (the attributes value is an array of attribute objects) and then appends the attribute to the JSON object.

To send the data to the server, the code needs to be in a URL format, so each key of the JSON object is looped through and the value is written out as property name = property value. This constructs the data string passed to the server as form data. Finally, the HttpClient.PostAsync method is set up to send the data to the server. For Open311 to post, Content-Type must be set to application/x-www-form-urlencoded, as it is in the code sample.

The postRequest function returns the HttpClient.PostAsync promise object the app can interact with further. After posting data to request.json results, the token value of the service request is received from the Open311 server. You can then use this token to look up the details for the request. By adding a then function to the postRequest function, you can build additional actions and workflows from the returned token value. As an example, the app might navigate to another screen when the token value returns or load the request data for the submitted request.

To complete the sample, register the postRequest function with the Data namespace and add the event handler code for btnSend in create.js, shown in **Figure 12**.

With all this input acceptance, one thing that's missing is validation. Always validate input to ensure you receive known good data from users. While I've omitted validation code here for brevity, it's included in the sample code. This code also uses the querySelectorAll function to find all the elements with a data-attribute-code attribute to easily access the dynamic input elements built-in to the ready function.

Gov 2.0: Build Trust Through Collaboration

Open311 is an excellent model for Gov 2.0 as local governments try to engage their communities in social issues. It illustrates how Gov 2.0 lets the community directly communicate with government through mobile technologies.

Simple reporting tools, with built-in opportunities to engage friends and other community members, lets users report issues and spread the word. By using a Visual Studio template, you can easily connect Open311 to an existing UI, helping to reduce development costs and speed to market. Windows 8.1 provides a wealth of opportunities for this app, such as using the Windows.Storage.Pickers.FileOpenPicker class to attach a media URL to the request, integrating the Search charm to allow users to find requests from anywhere, and quickly building rich interactions with Windows Azure. Open311 and some basic JavaScript skills can provide a deeply engaging experience for your community, enabling the community and its government to collaborate on important social issues. ■

Tim Kulp *leads the development team at United Healthcare International in Baltimore. You can find Kulp on his blog at seccode.blogspot.com or on Twitter at twitter.com/seccode, where he talks code, security and the Baltimore foodie scene.*

Thanks to the following technical expert for reviewing this article: *Eric Schmidt (Microsoft)*

Leverage Existing Web Assets as Data Sources for Apps

Frank La Vigne

We live in exciting times—we're experiencing a major shift from a Web-centric industry to one driven by mobile apps. This transition will redefine our industry and even the rest of the world. Despite this, many state and local governments have not yet begun producing mobile apps. However, most do have a Web site and some even have a social media presence.

Why are state and local governments so Web savvy, yet lagging in the mobile revolution? The answer may be complex, but likely all comes down to money and organizational inertia. Web technology is now a fairly mainstream and effective way to communicate with citizens. Given the wide deployment of Web hosting services, there's ample competition to reduce the costs of Web publishing. The same applies to social media. With hundreds of millions of users on Twitter and Facebook, even the most remote community can be assured that a fair share of its constituents can be reached.

Services and Mobile Apps

Along with widespread deployment of smartphones, the market has segmented to such a degree that developing a mobile app quickly escalates to a multiplatform development story with multiple codebases. Furthermore, many Web-savvy developers and architects have already moved to a service-oriented architecture (SOA).

SOA offers a number of benefits, such as separation of concerns and scalability. It also provides a nice platform-agnostic means of publishing data. The code on the mobile device becomes more about rendering the data or "painting the glass" for the consuming device. With the majority of business logic centralized on the server, multiplatform development becomes more manageable. Each client application needs only a basic codebase, making it simpler and more maintainable, as shown in **Figure 1**.

With these benefits in mind, why aren't more state and local governments adopting this approach? Once again, it boils down to budget. Quite often, the costs in terms of money, time and political determination are too high to reengineer systems and publishing processes. All this combines to make app creation a non-starter for this market segment.

Given these constraints, what you can do is leverage your existing Web assets and set up a façade that mimics the SOA approach. It may not be an ideal solution, but it can enable cash-strapped organizations to provide new mobile services to their constituents, as well as offer a roadmap for future enhancements.

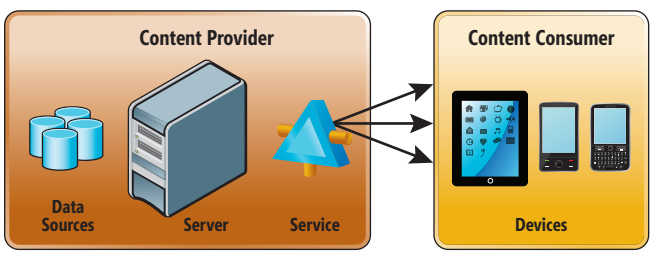


Figure 1 **Service-Oriented Architecture**

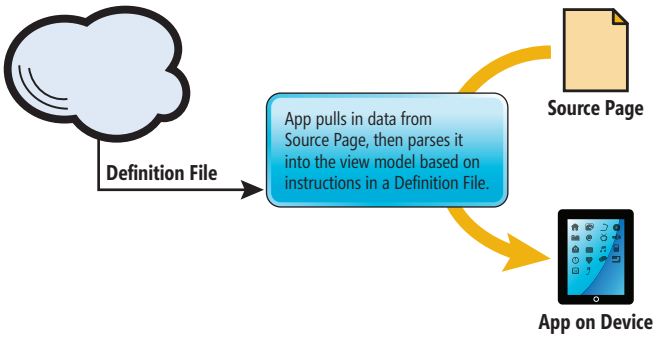


Figure 2 **Using a Definition File to Update an App**

A Common Scenario

Web sites are marvels of engineering. Using protocols and standards reaching back to the 1990s, Web site publishers can broadcast information cheaply and provide automated services to a worldwide audience at an extremely low cost. One way to repurpose Web-based content is simply to do nothing at all. Virtually all mobile platforms ship with a native Web browser. This means any Web site will be accessible on any mobile device with an Internet connection. There's no additional cost or effort to this approach—it's free.

However, many sites don't render gracefully on smaller screens and usability suffers accordingly. Fortunately, this can be mitigated by using CSS media queries. Like CSS, CSS media queries provide guidance on how content should render based on certain parameters, such as screen size and orientation. Best of all, you can add CSS media queries to your site and they'll apply only to browsers that understand them and be ignored by browsers that don't. Still, though CSS media queries can help bridge the gap between desktop and mobile screen sizes, this technology won't create native mobile apps.

Why Write an App?

The question today for cross-platform mobile developers is: Mobile-optimized Web site or native app? While developers debate which is better, business decision makers and stakeholders are listening to their customers and the market. Native apps optimized for a particular platform appear to be winning. Native apps have access to device-specific APIs, can access more on-device hardware, and can be written to take advantage of each mobile platform's aesthetic or design language.

Recently, users have been doing what they do best: behaving unexpectedly. While on their desktop, they'll consult a search engine such as Bing or Google. On their mobile devices, they'll look for apps. The people I see doing this tend to be both business decision makers and general users. When I ask why they behave differently on different devices, responses range from, "I don't know why, I just do," to, "App store results are more curated than search engine results," to, "It will look better on this screen."

> One way to repurpose Web-based content is simply to do nothing at all.

If these observations are indicative of the general user population, then Web developers are at the cusp of apps becoming the new Web site. Apps will be the must-have technology to engage with citizens in a more effective as well as cost-effective manner.

The Multiplatform Burden

One major challenge to making app development cost-effective is multiplatform support. In order to reach the widest audience, state and local governments must develop applications for Android, iOS, Windows Phone and BlackBerry devices. That means at least four different codebases to write, maintain and manage.

Private sector organizations generally pick one or two platforms on which to focus. State and local governments may not have that option because there could be regulatory requirements to make information accessible to everyone. Laws vary from locale to locale,

Debrief: Sourcing App Data from Existing Web Sites

Nearly every small town and county in the United States has a Web site, yet these communities rarely have apps. Part of this is the lack of a Web API and the money to build one. In this article, I discuss how to leverage existing Web assets and repurpose them for the era of the mobile app.

IT Brief:

As mobile apps increase in importance, the pressure for organizations to build them will increase. For apps to be useful, they need to contain relevant and up-to-date content. In most scenarios, this means exposing data via a robust Web-based API. For many cash-strapped smaller towns and counties, this represents a major obstacle. In this article, I demonstrate how to extract data from existing Web properties.

- Web pages present content in text-based HTML
- HTML on Web pages has structure
- The HTML structure can provide an app with real-time data with minimal effort

Dev Brief:

HTML is semi-structured data. It has internal structure that both presents and occludes information. This structure can be parsed into meaningful, discrete data elements via screen scraping.

- Create a Web API façade by screen scraping the HTML on your existing Web properties
- Definition files add resiliency and flexibility
- ShaZapp gives you a head start on creating an app that aesthetically and functionally aligns more to your brand and mission

More Information:

- ShaZapp: bit.ly/15im6lk
- ShaZapp Walk-Through Screencast: bit.ly/180bVmV
- Screen Scraper Utility Kit: bit.ly/ScreenScraperUtilKit

but cutting off citizens from information based on their choice of mobile platform seems like an unwise decision in the public sector.

It would be so much easier if more state and local governments implemented the SOA approach to their infrastructures. Rearchitecting and reengineering seem like unnecessary obstacles.

Another Way to Look at HTML

Nearly every state and local government entity has a Web site, each one supported by databases, content management systems and workflows. What if this existing infrastructure could be repurposed to serve mobile apps? What if these existing assets could have a façade layer placed around them so they could mimic the SOA approach?

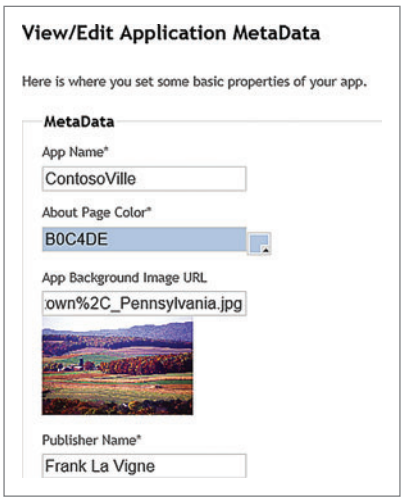


Figure 3 **Setting Basic Properties for Your App in ShaZapp**

Structured Data, Unstructured Data and Semi-Structured Data The industry has been dealing with structured data for a long time. Lately there's been a lot of talk about unstructured data. To many, unstructured data represents the final frontier of computer science. This is data that lacks structure, but has meaning and purpose. Faster processors, larger data stores and smarter, if not artificially intelligent, algorithms are needed to process this raw data into logical pieces of information.

> Nearly every state and local government entity has a Web site, each one supported by databases, content management systems and workflows.

In addition to structured and unstructured data, I propose a third data classification: semi-structured data. This is data with some internal structure that both presents and occludes information. HTML represents the simplest and most widely deployed form of semi-structured data.

Take a look at this sample HTML from a fairly typical HTML page (it's hard to call this "unstructured data"):

```
<table>
<tr><td width="30%">Events</td>
<tr><td><a href="events/julypicnic.htm">Fourth of July Picnic</a></td></tr>
<tr><td><a href="events/labor.htm">Labor Day Parade</a></td></tr>
<tr><td><a href="events/sept11.htm">Sept 11 Candlelight vigil</a></td></tr>
[...]
</table><br><br>
```

From this snippet of HTML, it's clear that HTML does contain structure. It may not be a structure you prefer, such as XML or JSON, but the sample clearly contains information that can be extracted. The question is how to extract it in a programmatic way that can be configured. The challenge is separating the HTML structures from the data.

There are a number of ways to do this: regular expressions, string matching, even CSS selectors. For my Screen Scraper Utility Kit (bit.ly/ScreenScraperUtilKit), I chose string matching. It's simple, performs well and is easy to maintain.

Screen Scraping?

Screen scraping—or the process of pulling down content from a display format, parsing it and then using it—generally has a bad reputation. Many consider it the option of last resort.

Why the reluctance to implement screen scraping? The objections generally fall into two categories:

1. Using a presentation layer as a data layer
2. Lack of resilience

Using one application's presentation layer as another's data layer is generally inferior design. However, in cases where there's no viable alternative, it can provide a bridge you can build rather inexpensively. As for screen scraping's lack of resilience, the problem is that should the source screen data change in any way, the reliant application will fail.

In this scenario, I have no Web API and no budget to build one. Screen scraping existing Web assets is the only viable way to extract data. As for the lack of resiliency, I have a solution for that.

Resilient Screen Scraping To make screen scraping more resilient, I added a definition file. Similar to the way antivirus software updates malware definitions, my definition file can change independently of the core parsing engine or the consuming app.

Figure 4 **The Definition File in XML**

```
<?xml version="1.0" encoding="UTF-8"?>
<Definitions IsReady="true"
  TargetUri="http://www.franksworld.com/msdn/Contosoville/">
  <Definition Name="Events">
    <StartMarkers>
      <Marker Value="<tr><td><a href="/>
    </StartMarkers>
    <EndMarkers>
      <Marker Value="</td>"/>
    </EndMarkers>
  </Definition>
  <Definition Name="Title">
    <StartMarkers>
      <Marker Value="">"/>
    </StartMarkers>
    <EndMarkers>
      <Marker Value="</a>"/>
    </EndMarkers>
  </Definition>
  <Definition Name="Link">
    <StartMarkers>
      <Marker Value="."/>
    </StartMarkers>
    <EndMarkers>
      <Marker Value="""/>
    </EndMarkers>
  </Definition>
</Definitions>
```

The definition file can reside at a location that can be updated independently of the application. Thus, when a source page changes, all you need to do is update the definition file. A user doesn't need to wait for a newer version of the app to get certified and published into the app store. The process is shown in **Figure 2**.

The Screen Scraper Utility Kit text-parsing engine takes raw text, primarily HTML, and uses text-pattern definitions from a Definitions object containing certain metadata and a list of Definition objects. Definitions have name attributes so they can be programmatically referenced.

Jump-Starting State and Local Government App Development

While the built-in starter templates in Visual Studio 2012 or later are a great place for you to start writing Windows Store apps, they don't necessarily help you to jump-start real-world app development. You'll still have to write code to connect to data sources. This can be particularly challenging for cash-strapped state and local government developers who don't have a Web API.

To help accelerate Windows Store app development for such organizations, our team developed ShaZapp. ShaZapp is an experiment in taking the screen-scraping model further, so resource-strapped communities can jump-start their Windows Store app efforts. ShaZapp lets less technically savvy users create a starting point from which developers can leverage their data already exposed on the Web. They can also use Bing images to create visually rich apps with minimal effort and cost.

Via the ShaZapp Web site, it's straightforward to generate the source code for a Windows Store app that's already tailored to your project.

Connecting with Citizens

Suppose you work for a small, rural town that wants to create an app to keep its citizens informed of local events. Your goal is to create an app that loads event data from the ContosoVille Web site, but displays the information within Windows Store apps. Let's assume this is your first Windows Store app and you may not be a full-time developer. You want to create an app that conveys the correct brand and messaging.

Creating the App Framework ShaZapp walks you through the process of creating an app with custom content and branding. It's designed to give an app builder a starting project geared more directly to her needs. The process consists of three steps: adding app metadata, creating data groups and generating the app source code.

The first step in creating an app with ShaZapp is naming the app, providing basic information about your organization, and setting some display and branding elements (see **Figure 3**). This lets you create a more customized project to work with in Visual Studio. It also fills in the required fields for getting your application published into the Windows Store.

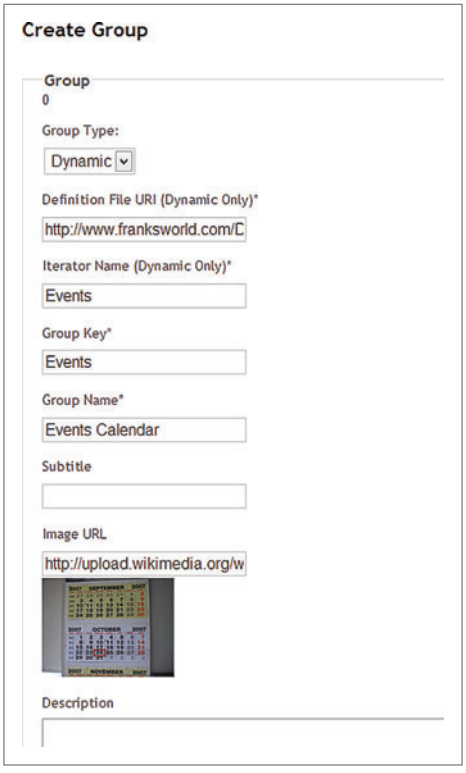


Figure 5 **Adding a Dynamic Group in ShaZapp**

Once these fields are filled in, you're ready to move on to the next step: creating data groups.

Creating Data Groups Now that your project has some basic information, the next step is to add data. All apps require data of some sort to be useful. ShaZapp provides two kinds of data group templates: static and dynamic. Static data groups contain data that remains largely unchanged. Good examples of this kind of data would be contact information for various city hall offices, trash pickup schedules and so forth.

Creating a static data group is straightforward. Simply click on Create New Group on the Groups page, then choose Static from the Group Type dropdown list. You'll notice right way some fields automatically disappear. The remaining fields all pertain to metadata about this data group. Only the Group Key and Group Name fields are required. Once all the field entries are complete, click on Create to create the group.

Next, add data items to the group. Click on Create New Static Item to bring up the Create New Static Item page. On this page, you can add data to the item. When you're done entering the data, click on create and you'll see this item in the static data group. Repeat this for all the data you'd like to include in the data group.

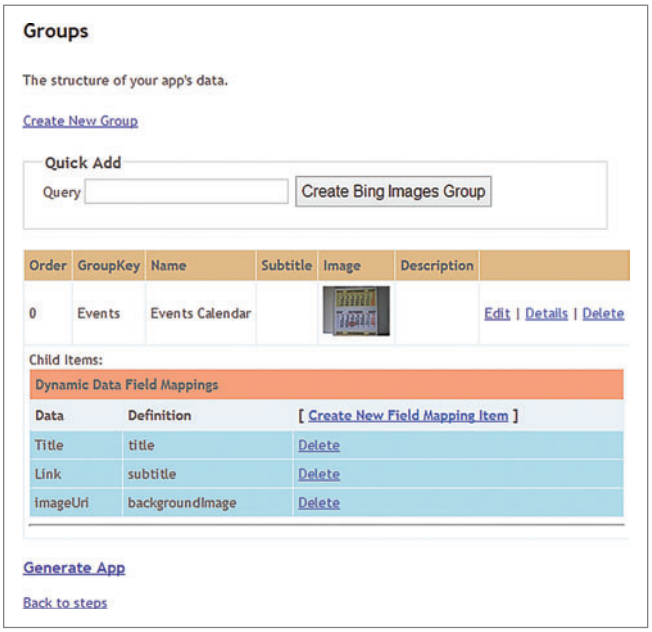


Figure 6 **Dynamic Data Field Mappings**

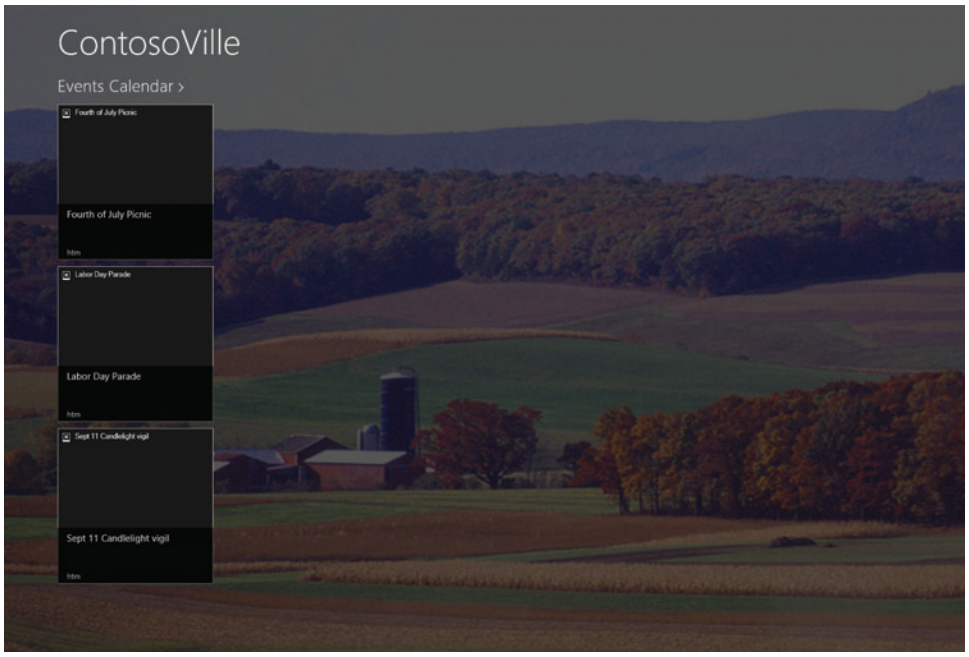


Figure 7 **The ContosoVille Windows Store App**

Pulling in Data Dynamically To add real value, you need to be able to pull in real-time data. Under ideal circumstances, you'd have a robust Web API to access the data. Not every organization has such an API. However, nearly every organization has a Web site. Web sites present HTML to an end-user browser. Often, the HTML has a consistent structure that you can leverage as a data source.

Both the Screen Scraper Utility Kit and ShaZapp use definition files that define start markers and end markers. Definition files are either in XML or JSON format and let the screen-scraper text parser know which character patterns define a starting point and end point for data embedded in the HTML.

To get started, I'll pull in the event's name and a link to the event detail page. The HTML is easy enough to parse. I've already created a definition file in XML (see **Figure 4**), and placed it on my Web server at bit.ly/1dkmEsA.

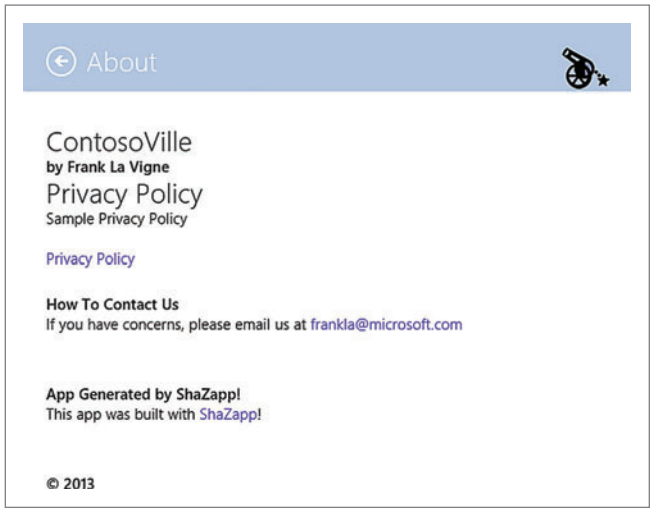


Figure 8 **The About Page for the ContosoVille Windows Store App**

First, I'll add a new group and make sure Dynamic is selected from the Group Type dropdown list , as shown in **Figure 5**. The Definition File URI field is where you reference the URI to the definition file. In Iterator Name, enter "Events." The remaining fields are display only and could have any number of values.

ShaZapp automatically adds default field mappings when adding a dynamic group. These fields are Title, Link and imageUri (see **Figure 6**). Normally, this is a nice time-saver, as these are common kinds of data. However, in this case, the source HTML doesn't contain an image associated with each event. As a result, I don't have an imageUri field definition. Be sure to remove this field mapping to avoid a runtime error in the final project by clicking on Delete.

Next, click on Generate App and then click on Generate Your App. Shortly thereafter, you'll see a download prompt for a zipped file. Download the file, extract the contents and then open the solution file. Once the solution is loaded into Visual Studio, run the project and you should see an app that looks something like what's shown in **Figure 7**.

Now bring up the Charms and click on About to bring up the About page shown in **Figure 8**.

> In mere moments you've created an app that displays data relevant to your purpose, instead of using a generic, out-of-the-box template.

This is where the metadata entered in ShaZapp gets placed in the generated project.

This app isn't perfect, but in mere moments you've created an app that displays data relevant to your purpose, instead of using a generic, out-of-the-box template. You also have the source code and can start modifying the app further based on your particular needs.

Examining the Generated Code

Take a closer look at that source code. In Solution Explorer, you'll see a structure similar to the output of a Grid Template project. However, if you open up the References folder, you'll notice this solution already has a reference to the Screen Scraper Utility Kit. If you look at the data.js file, you'll see further modifications, including

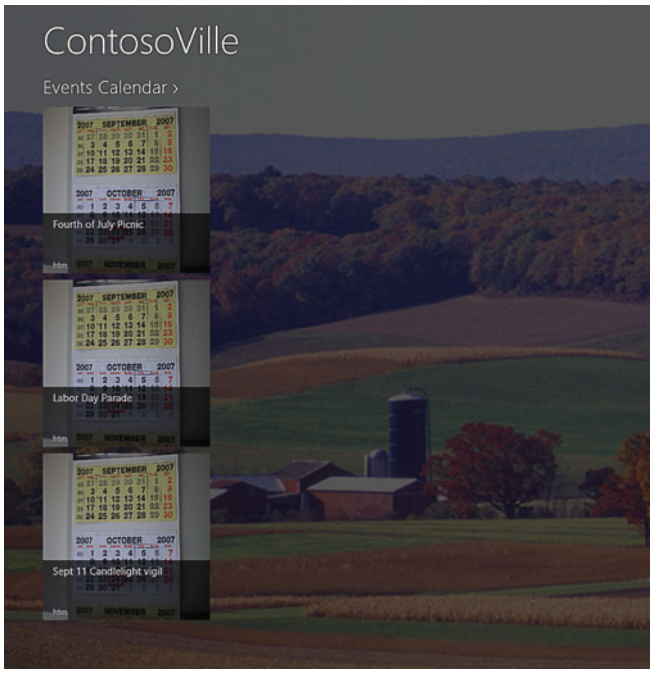


Figure 9 **Fixed Images in the ContosoVille Windows Store App**

common functions that interface with the Screen Scraper Utility Kit component. Following that code, you should see a section with two functions, loadEventsItems and parseEventsRawHtml. The loadEventsItemsfunction downloads the definition file and then calls the parseEventsRawHtmlfunction. This is where you can add an image to each of the events.

In the interest of brevity, I'll add the same image to each element, even though that's probably not the ideal end product:

```
var itemObject = {
  group: dataGroups[0],
  title: title,
  subtitle: subtitle,
  backgroundImage:
    "http://upload.wikimedia.org/wikipedia/commons/b/bb/WallCalendar.jpg",
  index: index
};
```

With that simple modification, I've removed the broken images and now have a slightly more polished product, as **Figure 9** shows.

What This Means

I took an existing HTML asset and converted it to a real-time, dynamic data source in just a few short steps. With some simple XML, the Screen Scraper Utility Kit can read and interpret the raw text of the Web page and convert it into a meaningful data model. Think of the power this gives smaller towns and communities.

Of course, the app as it exists now is hardly a finished product. There's more to be done in terms of parsing the data into more granular parts. Having the source code means you can continue to improve and customize the end product as you like.

Looking to the Cloud

Thus far, a service façade has been built on a device, an approach that can help state and local governments create Windows Store apps. How can you create a multiplatform solution that works for virtually any client? This issue is especially important to public-sector organizations, as they may have regulatory requirements to provide content to all citizens.

Instead of having the service façade on the Windows 8 device, what if you could move it to the cloud? Rather than duplicating the code across multiple platforms, it makes sense to perform the screen scraping at one central location and expose the extracted data structure at one REST service endpoint. Doing so would make the data pulled from the raw HTML accessible to all platforms that can understand a REST-based service (see **Figure 10**).

Moreover, it would give smaller communities a way to experiment with cloud computing with fairly low-risk data, meaning content that already resides on their public Web sites. This also provides a REST-based service architecture that can empower multiplatform development. From a developer's point of view, this architecture façade would be indistinguishable from an actual SOA implementation.

Wrapping Up

With the growing importance of mobile platforms, communities that lack native apps for the major mobile platforms will miss out on the opportunity to connect with their constituents. However, today's budget-crunched communities lack the funding to reengineer their back-end data systems to provide an ideal architectural pattern for multiplatform development. The short-term solution is to leverage existing Web-based assets and repackage them behind an on-device service façade. While screen scraping is rarely an ideal solution, the Screen Scraper Utility Kit makes the process easy, and it's hardy enough to adapt to changes in the HTML source data.

Looking forward, I hope to migrate my parsing engine to a Windows Azure-based model. This model puts the service façade into the cloud. With a mere change of a service endpoint, communities could easily swap out the mocked-up SOA with a real one once budgets allow for such development. ■

Frank La Vigne *is a technical evangelist for the Microsoft U.S. Public Sector DPE team where he helps public sector customers leverage technology in order to better serve their constituents. He blogs regularly at FranksWorld.com and recently started a YouTube channel called Frank's World TV (youtube.com/FranksWorldTV).*

Thanks to the following technical experts for reviewing this article: *Rachel Appel (consultant) and Roberto Hernandez (OverrideThis.com)*

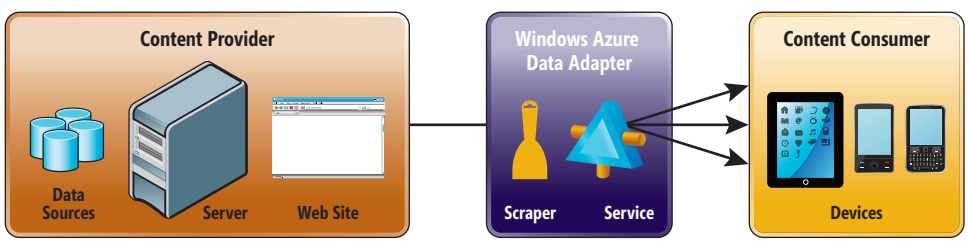


Figure 10 **Moving the Screen Scraper Utility Kit Service to the Cloud**

Enhance Citizen Services with Windows Phone 8 Wallet and NFC

Dale Michalk and Joel Reyes

City governments all over the world face tremendous challenges dealing with shrinking budgets while attempting to provide high-quality services to increasingly tech-savvy citizens. With enhanced mobile application capabilities and support for convenient payment and loyalty programs, Windows Phone 8 Near Field Communication (NFC) and Wallet functionality can help governments meet these challenges.

In this article, we'll take a look at the details of these technologies, as well as a sample app involving a fictional local transit system. The example presented here could easily be adapted to fit other scenarios involving justice, public safety, health care and other government services. You can download the sample app from archive.msdn.microsoft.com/mag201310gWallet.

Wallet and NFC Platform Elements

Windows Phone 8 provides a Wallet app and NFC capabilities that are separate yet complementary. The idea of a digital wallet is to free users from the burden of the concrete, non-virtual version, but without losing any of the functionality. The Windows Phone 8 Wallet can store a variety of cards—such as credit and debit cards, loyalty cards, membership cards, and coupons—and supports payment transactions. You can also track card balances and transactions, set up payment cards for purchasing apps and games, browse for local deals, and more. **Figure 1** shows the Windows Phone 8 Wallet hub app.

Windows Phone 8 NFC lets devices in close proximity share content such as photos or contacts. It can also establish a Bluetooth connection with something like a wireless speaker, for example. NFC standards deliver data packets between such devices using low-powered radio technology operating at 13.56 MHz. A range of mobile devices use NFC, including Windows Phone 8 and Windows 8 devices, wireless speakers, phone-charging stands, and payment terminals, as well as inexpensive NFC tags that can be embedded in or affixed to objects via stickers.

NFC tags can contain small amounts of data, such as URLs or map coordinates, that a device with an NFC antenna can read. NFC tags don't require their own power source because they're driven by the electro-magnetic field of the NFC device reading the tag. A common scenario could be NFC posters in transit stations that bring up a system map when the user taps on the poster.

The NFC Forum (nfc-forum.org) is the primary organization that standardizes communications between participating NFC devices. The most important specification for our purposes is the NFC Data Exchange Format (NDEF) that describes messages exchanged between NFC devices and tags. The specification defines a binary structure that can encode multiple records with different types of payloads. These are identified by headers for parsing the payload contents.

There are NDEF record types defined for a variety of uses. NDEF URI records work across platforms to open Web pages, or apps can

DOWNLOAD THE WINDOWS PHONE SDK TODAY

The Windows Phone SDK includes all of the tools you need to develop apps and games for Windows Phone.

bit.ly/UbFlDG

Code download available at archive.msdn.microsoft.com/mag201310gWallet.

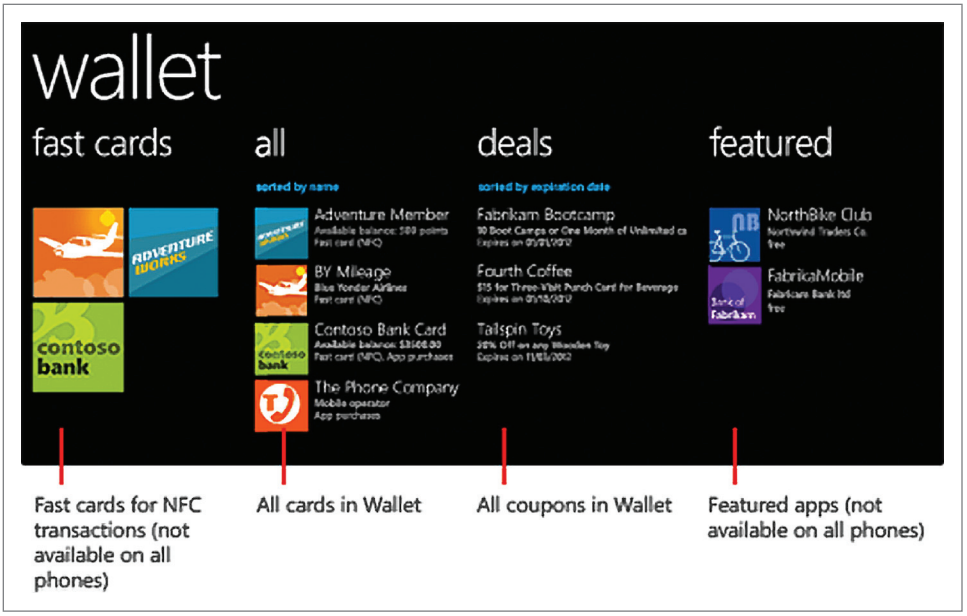


Figure 1 **Windows Phone 8 Wallet Hub App**

listen for a specific URI regardless of mobile OS. NDEF launching records can target specific platforms. The LaunchApp record, for example, is designed to invoke an application on the Windows Phone 8 platform. The NDEF record contains a GUID for a Windows Phone 8 app, as well as other parameters that provide context to the application launch, such as a transit station ID:

```
Windows.com/LaunchApp\tWindowsPhone&{1c3daa50-856d-46e1-bffb-
5d91f2768c88}\tstationid=5
```

Encoding multiple types of records on a single tag message lets a single tag support a variety of devices and capabilities without diminishing the UX.

Integrating an App with the Wallet

A Windows Phone 8 application can integrate with the Wallet API at different levels, as **Figure 2** illustrates.

A Wallet application on Windows Phone can create, read, update and delete (CRUD) Wallet items using the Wallet APIs provided in the Windows Phone SDK. A Wallet item will show up in the Wallet, and can have links that navigate to the application that created the item. You can also pin Wallet items to Start for fast access.

Wallet agents are specialized background agents an application can use to update Wallet items. They update Wallet data without direct execution by the owning application. This way, Wallet items can be updated with an application's supporting services when the user interacts with the Wallet hub.

NFC payment and NFC transactions with the Wallet require close collaboration between services providers, mobile operators and Microsoft. Drilling into the specifics of this collaboration is beyond the scope of this article.

The Wallet API

The core entry point to the Wallet API is the Wallet class, which has methods to retrieve or delete Wallet items. Several classes are derived from the Wallet Item class based on the type of information

Debrief: Windows Phone 8 Wallet/NFC

The ubiquity of mobile technology has changed our lives—as well as what we expect of government services. This article explores the digital wallet and Near Field Communication (NFC) technology built into Windows Phone 8, which can enable powerful new scenarios for citizen services.

IT Brief:

Technology decision makers and developers alike are responsible for finding new opportunities to reduce costs, increase revenue and enhance citizen engagement. They must also maximize current infrastructure investments. Windows Phone 8 is built with that in mind and lets IT managers conceive new solutions without sacrificing current investment in the Microsoft platform.

Wallet and NFC are capabilities built directly into the Windows Phone 8 platform, which allows the development of new apps reusing the same skills, tools and development platform you already have. People from all walks of life are already using mobile devices. IT planners now have the opportunity to deliver mobile city services that are engaging and profitable.

Windows Phone 8 Wallet and NFC can help city IT departments:

- Envision new delivery scenarios
- Increase citizen participation and engagement
- Accelerate and increase revenue
- Derive new value from current infrastructure investments

Dev Brief:

The Windows Phone Wallet API makes it simple to implement cards into any mobile app. The platform does the heavy lifting for the NFC plumbing, and the API exposes a simple proximity object model that abstracts all the underlying communication protocols. This means developers can stay focused on creating the best application features.

With the full programmability of the Wallet API and NFC, developers can:

- Take full control of Wallet items so they can be read, updated and deleted
- Create custom NFC tags by writing to the tags
- Leverage familiar skills and tools

More Information:

- Contoso Transit Sample App: archive.msdn.microsoft.com/mag201310gWallet
- General Wallet API: aka.ms/wp8-wallet
- Sample Code for Wallet Membership and Deals: aka.ms/w-membdeals
- Proximity for Windows Phone 8: aka.ms/nfcapidoc

Figure 2 **Integrating the Wallet API into a Windows Phone 8 App**

the Wallet item contains. For example, there could be items whose properties are used for deals and coupons, and those that represent cards with a transaction history. **Figure 3** represents the types used in a Wallet app.

OnlinePaymentInstrument handles payment cards, such as debit and credit cards. WalletTransactionItem handles cards used for membership, loyalty or other programs that can also store a list of transactions. Deal represents coupons or prepaid offers.

AddWalletItemTask is part of the Microsoft.Phone.Tasks namespace. This lets an application launch the Wallet application and display a dialog to save a Wallet item to the Wallet. The user can then choose to add the item to his Wallet.

Windows Phone 8 apps use a manifest file called WMAppManifest.xml to define an app's feature set. New capabilities have been added to Windows Phone 8 to support the Wallet.

ID_CAP_WALLET is the basic point of entry for using Wallet features for membership and loyalty cards and deals, such as the one we'll create in our sample application. Use IP_CAP_PAYMENTINSTRUMENTS and IP_CAP_WALLET_SECUREELEMENT with apps that require payment card or secure NFC functionality. We'll also enable ID_CAP_PROXIMITY, which supports the NFC features of the Proximity API. (You can also require NFC support from the Requirements tab by selecting ID_REQ_NFC.)

The Proximity API

The Proximity API is a cross-platform library for Windows Phone 8 and Windows 8 that supports a wide range of scenarios for establishing communications using proximity technologies. NFC tag support is a subset that centers on the ProximityDevice class. You can see a complete list of key methods in the ProximityDevice class at bit.ly/1avHJP4.

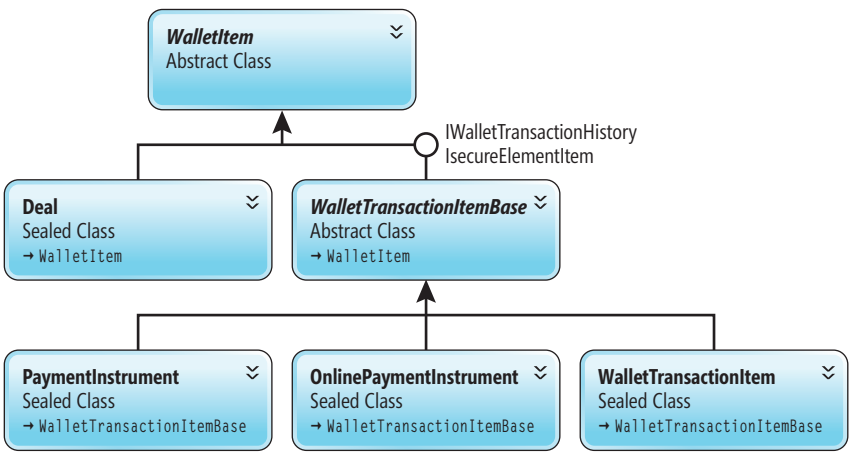


Figure 3 **Windows Phone 8 Wallet Object Model**

NDEF Library for Proximity APIs/NFC

The open source NDEF Library lets you easily parse and create NDEF records with the Windows Proximity APIs (NFC) for Windows 8 and Windows Phone 8. It extends the basic NDEF message support of the Proximity API with an object model for standardized NDEF record types, including the LaunchApp protocol for Windows Phone 8 apps that we use in our mobile Contoso Transit sample application. You can find the NDEF Library at bit.ly/XalLON or import it via the NuGet package manager in Visual Studio.

The creator of the NFC interactor (aka.ms/NDEFLibrary) contributed the NDEF Library to the development community. This excellent Windows Phone 8 app can not only create any NFC tag type, but also read and clone existing tags and dive into the guts of NDEF messages to assist in debugging any message format issues.

The Contoso Transit App

Now let's look at a sample transit agency application for the fictional City of Contoso that integrates with Windows Phone 8 Wallet and NFC functionality (see **Figure 4**). This mobile app offers basic informational services for a set of transit stations, including the arrival time of the next train. It also provides a loyalty card integrated with the Windows Phone 8 Wallet that tracks rider activities, lets users accrue points and offers deals for discounts with affiliated businesses.

The app's NFC functionality lets a rider tap an NFC tag located in a train station and launch the app to the appropriate station. If the user has the loyalty card enabled, the app will automatically check him in for loyalty points and deal offerings. If the application isn't installed on the device, it will prompt the user to download it from the store.

The sample solution consists of three Visual Studio projects:

1. Windows Phone 8 application project with the core application functionality
2. Windows Phone 8 library housing the mock Web service that represents the loyalty program back-end systems
3. Windows Phone 8 Background Task project with the code for the Wallet agent

Creating the Wallet Loyalty Card

The MainPage.xaml page is the application hub. This has links to navigate to stations, display or create the loyalty card associated with the app, or write out data to an NFC tag.

The application wires up a reference to the mock Web service for the transit agency back end in the constructor to display the stations. It also sets up an AddWalletItemTask object, which is key to creating a new wallet item after prompting the user:

```
public MainPage()
{
  InitializeComponent();

  service = new MockTransitService();

  addTask = new AddWalletItemTask();
  addTask.Completed +=task_Completed;
}

MockTransitService service;
WalletTransactionItem walletCard;
AddWalletItemTask addTask;
```

The Wallet.FindItem method is used to materialize a reference to the Wallet item representing the loyalty card, if it exists. If the card is found, we display the current balance, one of the properties available on the WalletTransactionItem class:

```
protected override void
OnNavigatedTo(NavigationEventArgs e)
{
  ... // Code removed for clarity

  walletCard = Wallet.FindItem("LoyaltyCard") as
    WalletTransactionItem;
  if (walletCard != null)
  {
    balanceTxt.Text = walletCard.DisplayBalance;
  }

  ... // Code removed for clarity
}
```

The code to add the Wallet calls the mock Web service to get the user's loyalty card data:

```
LoyaltyCardModel loyaltyCard = service.
GetLoyaltyCard("123456");
WalletTransactionItem item = new WalletTransactionItem("LoyaltyCard");
item.CustomerName = loyaltyCard.Name;
item.AccountNumber = loyaltyCard.UserId;
item.DisplayName = "Contoso Transit";
item.IssuerName = "Contoso Transit";
item.IssuerPhone.Business = "888-555-1212";
item.IssuerWebsite = new Uri("http://microsoft.com");
item.DisplayBalance = loyaltyCard.PointBalance + " points";
```

Here you can also include the code to add an image for the item.

If no card is found, the code invokes the AddWalletItemTask to prompt the user to create the card. Then it receives a callback to update the UI with the balance:

```
if (e.TaskResult == TaskResult.OK)
{
  walletCard = e.Item as WalletTransactionItem;
  balanceTxt.Text = walletCard.DisplayBalance;

  NavigationService.Navigate(new Uri("/CardPage.xaml", UriKind.Relative));
}
```

In the sample app, we preload the account so it already has a balance and a transaction. At that point, it navigates to the CardPage.xaml page, which shows the details. A real-world app would include a signup or authentication process on the front end to identify a user.

You can go to the Wallet app directly in Windows Phone 8 to see we created a card with all the details. You can also see the back link to the app that lets the user go back to the Transit app from the Wallet.

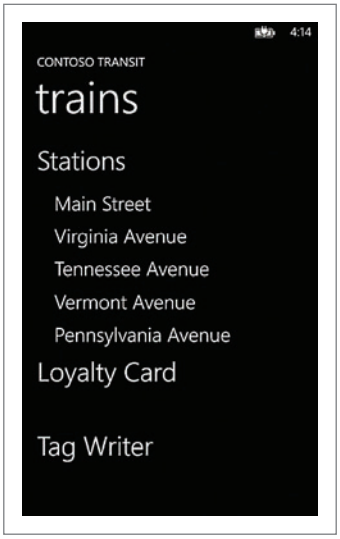


Figure 4 **The Contoso Transit App**

Station Check-In Function and Updating the Card

The StationPage provides details on station time, as well as a check-in function if there's a loyalty card enabled. The check-in button code updates the points balance after checking in with the mock Transit Web service and looks for a deal to add to the Wallet. This is a simple idea for rewarding users with points and deals that could easily be expanded for all kinds of interesting scenarios. Updating the WalletTransactionItem is as simple as updating its properties and calling its SaveAsync method, as shown in **Figure 5**.

The Deal class represents the coupons in the Wallet. It has a variety of properties that describe a wide range of deals. It also includes images so you can embed barcodes to support redemption systems with optical readers. Instead of having to create a deal with a special task, you simply instantiate the class and call its SaveAsync, as shown in **Figure 6**. The sample app prompts the user to determine whether to add the deal to his Wallet.

You can include code for adding bitmap images here as well.

Updating the Transaction History in the Wallet Agent

The Wallet Agent project shows how an agent updates Wallet items in the background. The key method is OnRefreshData, shown in **Figure 7**. This works with the mock Transit Web service to update the transaction details by creating WalletTransactionItem objects and adding them to the card's TransactionHistory dictionary. We use the datetime of the transaction to ensure the entries are in the collection and up-to-date.

Checking in at a Station with an NFC Tag

The sample's NFC functionality lets a properly formatted tag launch the app and navigate to a station using the LaunchApp protocol. Windows Phone 8 translates the parameters passed from this NFC record into a special-purpose URL parameter named ms_nfp_launchargs. You can identify this in the startup page of your app, as shown in **Figure 8**.

Figure 5 **Updating a Loyalty Card in the Wallet**

```
private async Task UpdateLoyaltyCard(int stationId)
{
  int balance = service.UpdatePointsOnCheckIn(walletCard.AccountNumber, stationId);

  walletCard.DisplayBalance = balance + " points";

  DealModel deal = service.GetDealForStationCheckIn(stationId);
  if (deal != null)
  {
    string prompt = String.Format("A deal is available for {0} from {1}.
      Add to wallet?", deal.Name, deal.Issuer);
    MessageBoxResult result = MessageBox.Show(
      prompt, "Coupon available", MessageBoxButton.OKCancel);
    if (result == MessageBoxResult.OK)
    {
      await AddDealToWallet(deal);
    }
}

  await walletCard.SaveAsync();
}
```

Figure 6 **Adding a Deal to the Wallet**

```
private static async Task AddDealToWallet(DealModel model)
{
  Deal deal = new Deal();
  deal.DisplayName = model.Name;
  deal.IssuerName = model.Issuer;
  deal.StartDate = DateTime.Now;
  deal.ExpirationDate = DateTime.Now.AddDays(model.DaysValid);
  deal.IsUsed = false;
  deal.Code = model.Code;
  deal.MerchantName = model.Issuer;
  deal.MerchantPhone.Business = model.MerchantPhone;

  await deal.SaveAsync();
}
```

Figure 7 **The OnRefreshData Method**

```
protected override async void OnRefreshData(RefreshDataEventArgs args)
{
  MockTransitService service = new MockTransitService();

  foreach (WalletItem item in args.Items)
  {
    WalletTransactionItem walletCard = item as WalletTransactionItem;
    if (walletCard != null)
    {
      LoyaltyCardModel loyaltyCard = service.GetLoyaltyCard(
        walletCard.AccountNumber);
      if (loyaltyCard != null)
      {
        foreach (var scan in loyaltyCard.StationsScanned.Keys)
        {
          if (!walletCard.TransactionHistory.ContainsKey(scan.ToString()))
          {
            StationModel station = loyaltyCard.StationsScanned[scan];
            WalletTransaction txn = new WalletTransaction();
            txn.TransactionDate = DateTime.Now;
            txn.Description = station.Name;
            txn.DisplayAmount = station.Points + " points";
            walletCard.TransactionHistory.Add(DateTime.Now.ToString(), txn);
          }
        }
      }
    await walletCard.SaveAsync();
    }
  }
  NotifyComplete();
}
```

Figure 8 **The ms_nfp_launchargs Parameter**

```
protected override void OnNavigatedTo(NavigationEventArgs e)
  {
    String launchArgs;
    if (NavigationContext.QueryString.TryGetValue(
      "ms_nfp_launchargs", out launchArgs))
    {
      if (launchArgs.Contains("stationid="))
      {
        string stationId = launchArgs.Substring(
          launchArgs.IndexOf("stationid=") + 10);

        NavigationService.Navigate(new Uri(
          "/StationPage.xaml?checkin=true&stationid=" +
          stationId, UriKind.Relative));
        NavigationService.RemoveBackEntry();
      }
    }
... // Code removed for clarity
```

Figure 9 **The SubscribeForMessage Method**

```
private void writeBtn_Click(object sender, RoutedEventArgs e)
  {
    station = stationLst.SelectedItem as StationModel;
    if (station != null)
    {
      if (device != null)
      {
        writeBtn.Visibility = System.Windows.Visibility.Collapsed;
        device.SubscribeForMessage("WriteableTag", TagLocated);

        statusTxt.Text = "Tap tag you wish to write";
      }
      else
      {
        MessageBox.Show("Unable to initialize NFC hardware");
      }
    }
    else
    {
      MessageBox.Show("Please select a station for the tag info");
    }
  }
```

Figure 10 **Creating and Publishing an NDEF Message**

```
private void TagLocated(ProximityDevice sender, ProximityMessage message)
{
  device.StopSubscribingForMessage(message.SubscriptionId);

  string appId = Windows.ApplicationModel.Store.CurrentApp.AppId.ToString();
  string stationArgs = "stationid=" + station.Id;

  NdefLaunchAppRecord launchAppRecord =
    new NdefLibrary.Ndef.NdefLaunchAppRecord();
  launchAppRecord.Arguments = stationArgs;
  launchAppRecord.AddPlatformAppId("WindowsPhone", "{" + appId + "}");

  NdefMessage msg = new NdefMessage { launchAppRecord };

  device.PublishBinaryMessage("NDEF:WriteTag",
    msg.ToByteArray().AsBuffer(), TagWritten);

}
```

We parse the message to get the appropriate station ID, and then navigate there with a special parameter the station uses to automatically check in the user. If a user taps a tag, the device will prompt him to confirm that he wants to handle the tag contents with the Contoso Transit app.

Writing an NFC tag

To test the NFC scenario, the app contains a small writing page named WriteTag.xaml. This shows how to use the Proximity API for locating and writing to a tag. The first step is to get the default ProximityDevice representing the NFC hardware on the phone in the page constructor:

```
public WriteTagPage()
  {
    InitializeComponent();

    device = ProximityDevice.GetDefault();
    service = new MockTransitService();
  }
```

When the end user selects the station to which he wants to write, we call the SubscribeForMessage method to find a writeable tag, as shown in **Figure 9**.

Once the tag is located, it's just a matter of creating the NDEF message using the NDEF Library NdefLaunchAppRecord and publishing the message using ProximityDevice method PublishBinaryMessage, as shown in **Figure 10**. The app Id GUID is harvested from the phone's application model API. You can also find it in the WMAppManifest.xml file.

Wrapping Up

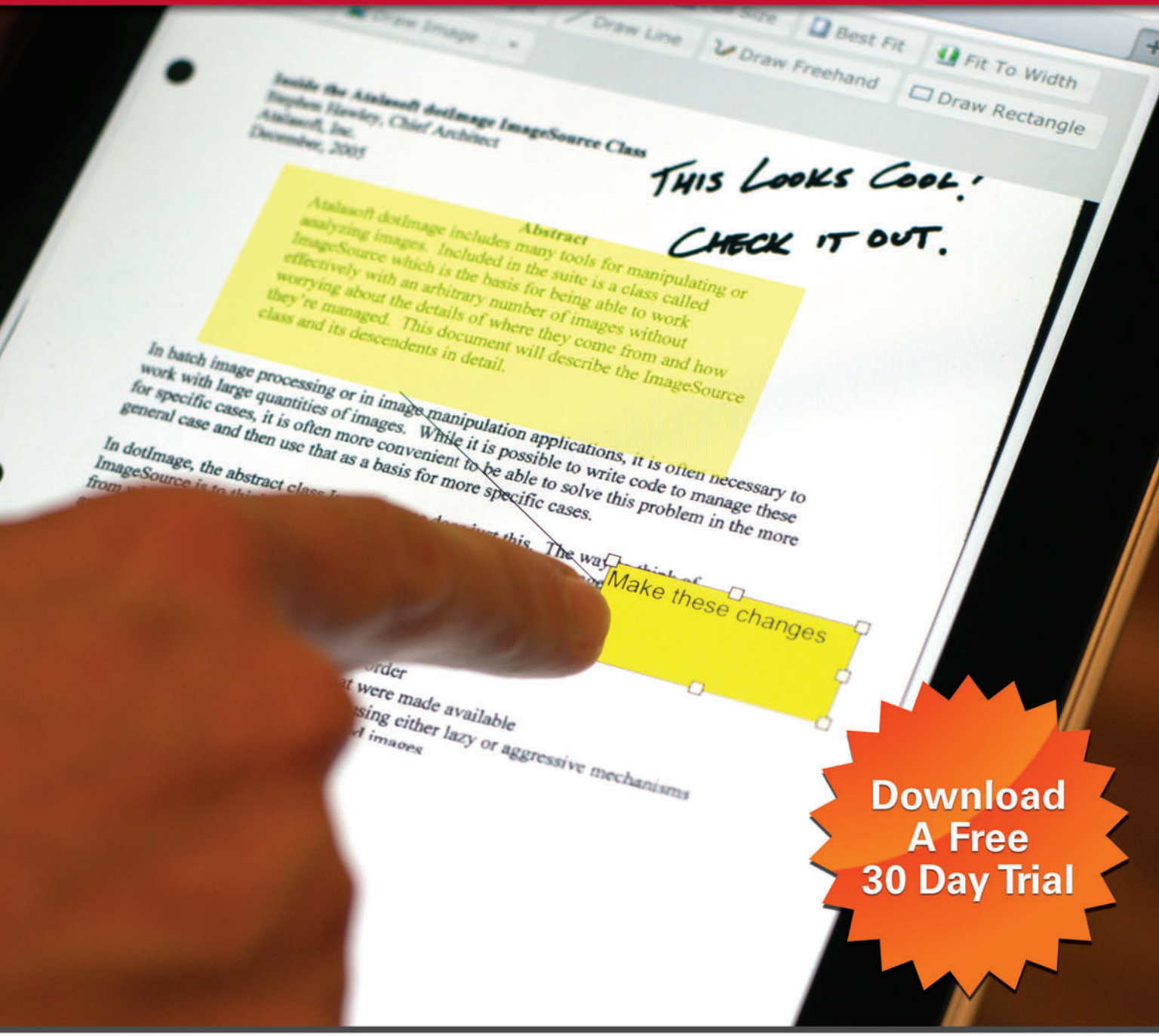
Windows Phone 8 provides a great platform and API. They ease the burden of building mobile solutions that involve digital wallets and NFC technologies. This example illustrates one way of using them to improve the quality and efficiency of city services in a transit agency. You could easily adapt the same techniques to other government solutions including health care, public safety and other areas of citizen-service delivery. ■

Dale Michalk *is a solution achitect for Microsoft Public Sector.*

Joel Reyes *is a startup/citizenship evangelist for Microsoft Public Sector.*

Thanks to the following technical expert for reviewing this article: *David Gardner (Microsoft)*

Build A Mobile Document Viewer with Annotations, Touch Interfaces, Zooming, Pagination, and More



Download A Free 30 Day Trial

Atalasoft
A Kofax Company

.NET Web Scanning & Imaging SDKs

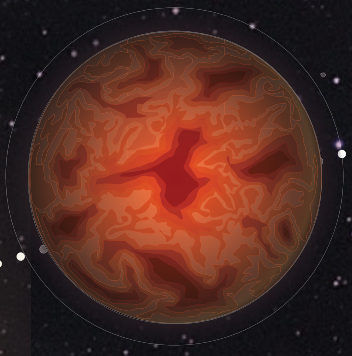
1.413.572.4443 ◆ www.atalasoft.com

Visual Studio® LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

2014 Dates Announced

Much like outer space, the .NET development platform is ever-changing and constantly evolving. Visual Studio Live! exists to help guide you through this universe, featuring code-filled days, networking nights and independent education. Whether you are a .NET developer, software architect or a designer, Visual Studio Live!'s multi-track events include focused, cutting-edge education on the .NET platform.



LAS VEGAS

LIVE! 360 DEV

COMPREHENSIVE TRAINING FOR THE DEVELOPER WORLD

LIVE! Visual Studio | LIVE! SQL Server | LIVE! SharePoint | LIVE! WebDev | LIVE! Modern Apps

Visual Studio Live! Las Vegas

Part of Live! 360 Dev

March 10 – 14
Planet Hollywood
Las Vegas, NV

The Developer World is always expanding; new technologies emerge, current ones evolve and demands on your time grow. Live! 360 Dev offers comprehensive training on the most relevant technologies in your world today.

Visual Studio Live! Chicago

May 5 – 8
Chicago Hilton
Chicago, IL

Building Government Business Applications with Microsoft Dynamics CRM

Marc Schweigert and Andrew Schultz

When developers evaluate technologies for building government business applications, Microsoft Dynamics CRM may not be the first to come to mind—but maybe it should be. Build your application on Dynamics CRM and experience shortened time-to-value because of the many platform features that limit the amount of code you'll need to write. Enjoy out-of-the-box enterprise scalability, reduced maintenance and enhanced agility for business users.

A significant and increasing number of organizations across the U.S. federal, state, and local governments, as well as the health-care and education verticals, have been doing just that. They've been using Dynamics CRM as a custom development platform to solve business problems. The solutions they've built vary widely. Some examples include:

- Planning mission and workforce readiness for the military and national security agencies
- Ensuring veterans receive benefits
- Tracking prison inmates
- Managing government benefits and assistance coordination
- Managing student retention in education
- Coordinating care for people with developmental disabilities

While these solutions span a variety of use cases, they're united by a set of common characteristics that Dynamics CRM is uniquely qualified to provide. So let's start with you—because chances are, as a person tasked with building a government business application, you need to produce an application with these same characteristics.

This article discusses a prerelease version of Microsoft Dynamics CRM. All information is subject to change.

Your Problems

- **UI:** Your application needs to give users data-entry forms that represent relational data (people, places and things).
- **Business Logic:** Your requirements demand data validation, complex logic and algorithm execution, and so forth.
- **Business Process Automation:** You're tasked with automating processes that span long periods of time and might have complex steps. These processes previously relied on human effort to complete.
- **Business Intelligence (BI):** You've been asked to produce reports from your application's data that yield insights that improve your organization's performance.
- **Enterprise Scalability:** Your application will be deployed to a pilot group of users at first, but eventually it will host thousands of users. You need to be able to handle growth from a small infrastructure of one or two servers to a large server farm over time. Buying all the hardware right now isn't an option.
- **Document Management and Storage:** Your organization projects that paper forms will play an important role in operations for the foreseeable future, but it wants to store the forms digitally. Additionally, other forms exist that can be filled out digitally, but can't be converted to Web forms over a database, so you need to store the document in its entirety.
- **Granular, Role-Based Security:** Security is important in the government. You need to ensure that when User A logs in, he can see or do certain things, and when User B logs in, she can see and do different things.
- **Mobility:** The end users of the system have been begging for something mobile so they can move through the office with the people they're helping, without having to log into different PCs at various locations.

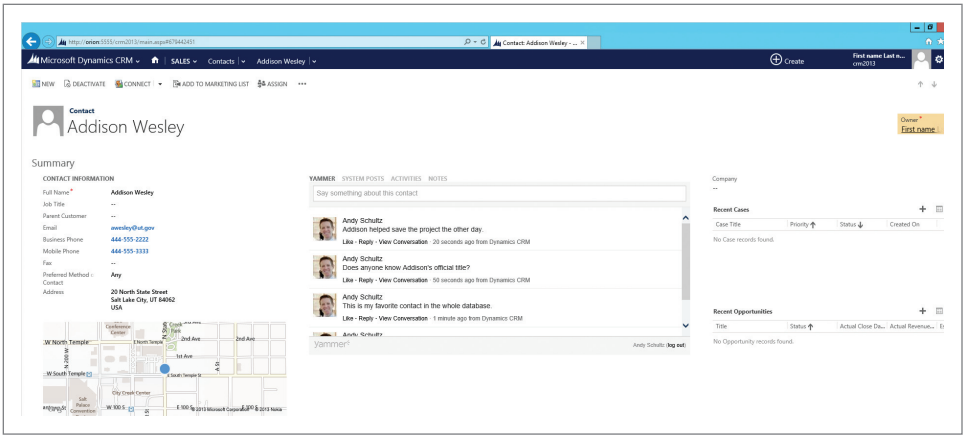


Figure 1 **The Microsoft Dynamics CRM UI**

- **Integration with External Systems:** Admit it—you've got legacy systems. There's still important data in these other systems, and you're not going to migrate it out all at once. Your application needs to be capable of integrating with other systems to share, move or view data.

So, what does this have to do with customer relationship management (CRM) and salesforce automation? Let's talk about CRM, and about how you're going to accomplish all of this with Microsoft Dynamics CRM.

Your Platform

First, let's look at *why* Dynamics CRM is capable of meeting the unique requirements of your application—because it certainly did start out as an application designed to manage a company's sales, marketing and customer service data. But there are as many different ways to do these activities as there are companies that do them. So Microsoft built Dynamics CRM with the ability to flex and meet various customer needs.

Moreover, the efforts of these departments are rarely carried out in isolation. There are other departments doing related activities, regularly moving in and through the work stream of the sales, marketing and service departments. Customers want to track this data as well, but it doesn't fit in the enterprise resource planning (ERP) applications being used. So Dynamics CRM provides the ability to be even more flexible. In the end, this flexibility enables data to be used for purposes that have nothing to do with sales, marketing or service. Using Dynamics CRM to address these custom use cases is typically referred to as Extended CRM. More commonly, the community of people who build these solutions affectionately call it XRM.

We'll discuss the ways Dynamics CRM can be extended later in this article, but its core customization feature is its extensible data schema. Without writing code, you can create custom data objects (called entities) and attributes, and create custom relationships between the entities. These entities represent people, places, or things, similar to the types or objects you'd create in code. Every entity created has one or more forms for data entry, and data grids for viewing and filtering lists of entity records. These forms and grids are automatically included in the security model, allowing them to be configured with role-based security. Charts can be created and added to custom dashboards that visualize data about the entity for users. Finally, all of these customizations can be packaged in an exportable container called a "Solution," which can then be archived or imported into other deployments of Dynamics CRM.

Now that you've been briefed on Dynamics CRM, let's see how the requirements can be met for your application.

UI

You need a customizable, attractive front end for your users. Let's face it—a lot of the applications users see are ugly enough to make them pucker. Giving them an application that looks cool might

Debrief: Windows Azure Security

Microsoft Dynamics CRM is being used by government organizations in the United States and across the world to build business applications. This article discusses why Dynamics CRM is a great platform for government business application development.

IT Brief:

The cost of building a business application from scratch is likely to be greater than using Dynamics CRM as a platform. Consider the following benefits:

- Shortened time to value because of the many platform features that limit the amount of code needed.
- Enterprise-ready with security and scalability.
- Maintenance and modernization of the platform performed by Microsoft.

Dev Brief:

Dynamics CRM is a platform for developing custom business apps. Developers can accomplish many tasks that would have previously required intensive code with simple configuration in Dynamics CRM, and focus code-writing efforts on solving more complex business challenges.

- Security is part of the platform.
- Forms and grids in a modern UI come standard, with no programming code.
- Programming code can easily achieve complex business logic, integration with other systems and customization of Dynamics CRM forms.

More Information:

- Dynamics Government Showcase (login required): bit.ly/1adw0Fr
- Posts from the Dynamics CRM Blog relating to SharePoint: bit.ly/19qXE0b
- A Primer in Extended CRM Development: bit.ly/VqR0r5

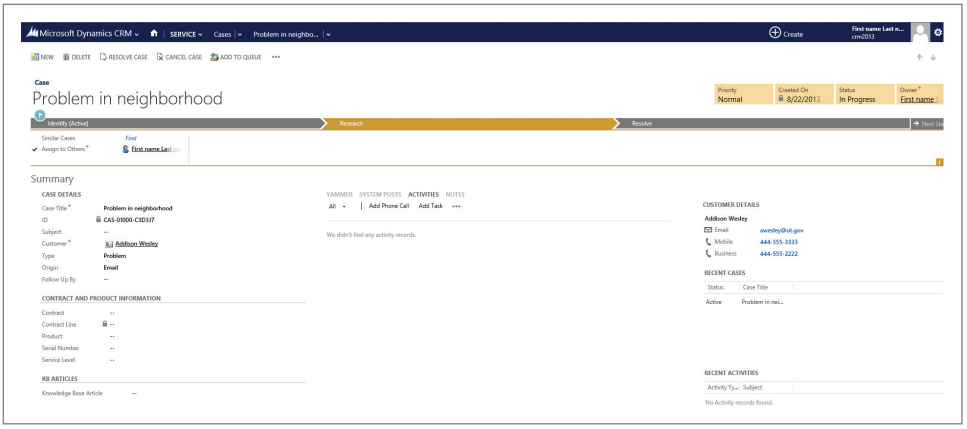


Figure 2 **A Dynamics CRM Record with a Process Flow on the Form (the Process Flow Is the Area Immediately Below the "Problem in neighborhood" Title)**

change their lives, save their marriages, improve their neighborhoods and stop global warming.

The UI of Microsoft Dynamics CRM, shown in **Figure 1**, is Web-based and built on ASP.NET. As with all ASP.NET applications, the CRM Web server sends HTML, JavaScript, CSS and images to the browser that renders the UI. Whether you're using Dynamics CRM through a Web browser or the Outlook add-in, the majority of the UI you see is HTML-based, with the exception of a few places in the Outlook add-in. However, even in the add-in, objects such as forms are simply surfaced in Outlook through a browser frame. In offline mode, those Web pages are served through a lightweight Web server running on the client computer.

The primary mechanism for extending the Dynamics CRM UI is something called a "Web resource" (for an example, see the map control in **Figure 1**). Web resources are simply HTML, JavaScript, CSS, images or a few other endorsed file types that a developer can upload to Dynamics CRM. After you upload Web resources, Dynamics CRM will serve HTTP requests for those files from the Dynamics CRM Web servers.

Dynamics CRM exposes both SOAP and REST Web services that can be called from JavaScript. The SDK contains a series of samples and helper libraries to help you do so. Because you're working in good old HTML/JavaScript/CSS, by calling Web services you can pretty much do anything that's possible with those technologies in the context of your XRM application.

However, there is one caveat. The rule of thumb with Dynamics CRM—or with any server product that produces HTML and allows for extending the UI—is that you may not modify the UI the server sends to the browser using HTML Document Object Model (DOM) manipulation or libraries such as jQuery. However, if you created the HTML, you can perform DOM manipulation all you want. The CRM SDK covers this in more detail. The SDK also provides a JavaScript library, typically referred to as Xrm.Page, which is a layer of abstraction on top of the HTML UI that enables, for example, interacting with controls and data on Dynamics CRM forms.

HTML5 So what about HTML5? As long as the browsers rendering the Dynamics CRM UI are HTML5-capable, your applications can take advantage of it. Because you can upload JavaScript files to the server, you can deploy popular third-party JavaScript libraries to Dynamics CRM as well.

Once you understand that the core UI extensibility of Dynamics CRM is just HTML/JavaScript/CSS together with Web service calls to interact with the server, *and* you understand that Web resources are just those assets uploaded to the server and served back to the browser through the same Web servers that serve the CRM UI, all sorts of "aha!" moments about how you can extend the CRM UI begin to occur.

Note that Web resources are protected by the various authentication mechanisms supported by Dynamics CRM. As a result, they won't be served by the Web server unless a user has logged in.

To bring together all the concepts discussed so far, consider the following example:

Imagine a completely customized jQuery Mobile-based UI that uses HTML5 geolocation APIs to get a user's latitude and longitude, then stores that location as a record of proof that a field worker was where he was supposed to be at the time of a field service appointment. The field worker's simple task would be to navigate to a URL using a browser favorite and then log in. That URL would be, as you might have guessed, pointing to an HTML Web resource that provides a completely custom UX using HTML/JavaScript/CSS. After the user logs in, the UI calls the CRM Web services to retrieve and save data. On the CRM platform, all custom business logic, workflows, and role-based security that had been configured would be executed or enforced in response to any Web service requests. This example, though simple, demonstrates the power of using Dynamics CRM as a platform to build business applications.

Business Logic

There are multiple tools in Dynamics CRM for designing business logic. Your requirements include data validation, complex logic and algorithm execution. Let's talk about some tools you can use to knock this one out of the park.

Plug-Ins Plug-ins are server-side logic, written in C#. They execute in relation to database transactions, and can be configured to execute before or after a transaction. They can be configured with filters that keep them from running when the attributes of an entity being passed to the database don't match the desired conditions, so that they don't cause too much overhead. Plug-ins provide an extensive set of capabilities, allowing access to almost everything in the SDK.

Actions Actions are new in Dynamics CRM 2013, which is slated for release this fall. Before Actions, you couldn't execute arbitrary server-side business logic—you could only write server-side business logic that reacted to events related to Dynamics CRM entities. Actions are a way to define arbitrary business logic. You can think of Actions as a declarative way to author the equivalent of a

function or method call, complete with input and output parameters. Actions are exposed through the Dynamics CRM SOAP Web service. Therefore, they're callable from Web resources, other Dynamics CRM extensibility points that can execute custom code and outside callers. Common scenarios in which Actions might be used include "Escalate," "Approve" or similar activities.

Actions have an added bonus in that they enable a business analyst to compose server-side business logic without writing code. They're written using the same designer used for authoring workflows. Therefore, all facilities available to the workflow designer, including custom workflow activities, are available for Actions. Actions are a powerful new extensibility point in Dynamics CRM 2013 that are sure to be leveraged in interesting ways.

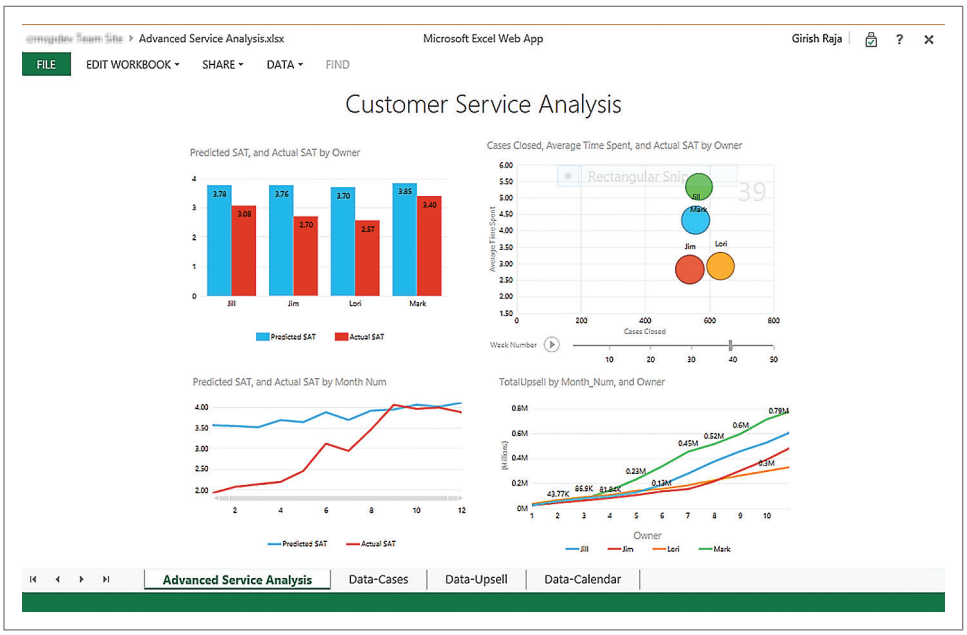


Figure 3 **Example Power View Report Showing a Scatter Diagram with a "Play Axis" at the Top Right**

Workflow Another configuration that a business analyst can create without code is a workflow, a server-side business logic routine that can be triggered manually by a user, or by a system event such as the update or create event associated with a type of record. Dynamics CRM 2011 allowed only for asynchronous execution of workflows. As a result, there were lots of areas in which code was required to solve a problem if the logic needed to be executed synchronously. Dynamics CRM 2013 adds support for synchronous execution of workflows, which will have the net effect of reducing the need for custom code.

Portable Business Logic Finally, business analysts can declaratively configure business logic that executes on the Web forms for Dynamics CRM records. Earlier versions of the product required JavaScript to accomplish these actions, and a much broader set of actions is still available for those willing to write script. However, the following key capabilities represent some of the most commonly implemented logic on forms:

- Validate data entry and throw errors
- Perform simple calculations (+, -, *, /)
- Show/hide fields
- Enable/disable fields
- Set field values
- Make fields required or recommended

The ability to perform these actions without having to write code reduces the amount of script needed to deploy many solutions. In addition, by empowering business analysts to create this type of logic without the assistance of developers, Dynamics CRM helps customers be more agile and able to quickly support rapidly changing business needs with technology solutions. Despite the high value of this type of feature, it would rarely pass the cost/benefit analysis and be included in plans for an application developed from the ground up. With Dynamics CRM, this functionality is already part of the platform.

Business Process Automation

You need to automate a process whose inefficiency is wasting millions of dollars every year. Let's talk about a no-code solution that can help your people follow the process and collect the right information as they do so: process flow. The process flow allows the person doing the configuration to create stages in a process for a type of record, such as a customer service case. The person can then put different fields in the stages, declare whether the fields are required before moving on to the next stage, and thus ensure that end-users consistently follow the correct process and collect the correct data. The end result is what you see in **Figure 2**.

With the other types of business logic customizations we've mentioned, you can run algorithms, more complex logic, and even reach out to get data from or push data to other applications via Web services when certain parts of the stage are completed.

Business Intelligence

So you have these requirements to provide certain key data in reports. Were you to build this application from the ground up, you'd have choices about how to meet this objective, but none of them would be trivial. Using Dynamics CRM as the platform for your application, you also have choices, and some of them are trivial.

A presenter in a session on Dynamics CRM in health care recently made the point that while we're all intrigued by the concept of big data, we shouldn't overlook the fact that what most end users really need is small data to help them make everyday decisions. Whether your BI game plan involves using data from a data service, gathering ambient data from sensors to combine with the transactional data in Dynamics CRM, or simply organizing insights

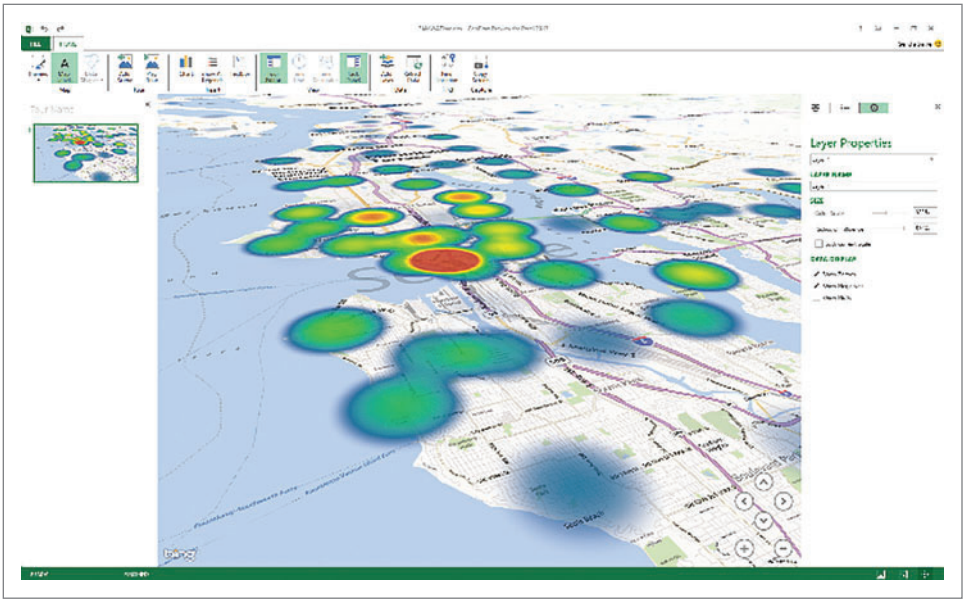


Figure 4 **A Sample Power Map Report**

from the data you track in your application, there are several viable ways to get the data out of its transactional resting place and into more dynamic, visually appealing formats for deeper analysis. Plenty of material already exists that covers the standard Dynamics CRM BI experiences, which include building SQL Server Reporting Services (SSRS) reports, charts and dashboards.

Rather than describe these standard capabilities in this article, we'll focus on some of the lesser-known ways you can leverage Dynamics CRM and surrounding Microsoft technologies to allow for deeper analysis, in particular the Microsoft Office tools that allow you to perform self-service BI—Power Pivot, Power View, Power Map and Power Query.

Microsoft Office is perhaps the most powerful companion to Dynamics CRM. The integration with Microsoft Outlook is the most prominent aspect of this, but integration between Dynamics CRM and Office is far more pervasive than this single feature. The BI tools discussed hereafter are made possible via recent advances in Excel data visualizations. Together with the "Export to Excel" feature that has been in Dynamics CRM since early versions, these features enable a level of visual data analysis that wasn't previously possible.

Power Pivot An Excel add-in that allows you to combine data from different sources into one data model in Excel. This in-memory data-processing tool can work through huge volumes of data for analysis without showing any sign of lag. The ability to analyze so much data so quickly, combined with the rapidly increasing availability of public and enterprise data, opens the door for some interesting data exploration and analysis.

Power View A data-visualization tool that runs inside of Excel. In a process similar to building pivot tables and pivot charts, you can organize data into axes to create beautiful reports that make trends in the data easier to spot.

Power Map Similar to Power View, except the visualizations are all map-based. When your data has a geographical dimension, throw it into Power Map and see the data laid out across geographical space.

Power Query A data-import tool right inside of Excel that can pull data out of XML, JSON and OData Web APIs. It can even rip data out of HTML tables in Web pages. If there's any supplemental data that you want to sprinkle your Dynamics CRM data with for greater insight, Power Query can help.

Why are we talking about Excel so much in an article about Dynamics CRM? Because when you build your government business application on Dynamics CRM, you can get data from your application to Excel with the push of a button. And with the ability of Excel to refresh its data from Dynamics CRM, you can build your Excel BI tool once and get new data from your application every time you open it. You can even take the Excel file and put it in your application's reports area, so your users can open the file right from the Dynamics CRM interface.

Let's get down to details. Imagine your agency requires you to prepare a report that shows the satisfaction levels of the customers of the agency's field offices. Maybe the agency just asks for a static report, or maybe it wants the ability to explore the data. In either case, you decide to over-deliver on this requirement, because it's so easy to provide more information. Not only will your agency have a report it can explore a little bit, it'll also have a full self-service BI tool that will give it insights it never thought possible.

So how do you start implementing this solution? You'll need to get your transactional data schema set up in your application, which is also a prerequisite for users to start generating data. Once that data exists, you'll export it to Excel.

Once you have the data in Excel, you're free to do whatever you want using the tools listed earlier. Use Power View and Power Map to lay the data out in visually stunning formats that yield much more nuanced insights than a table or simple chart.

For example, one of the most interesting and insightful data visualizations available in Power View is the scatter diagram. In addition to the X and Y axes and the bubble-size axis, a scatter diagram provides a "play axis" that contains a dimension by which the measures will be animated when you push the Play button. The most obvious data to use in this axis is time. Put an appropriately scaled set of time values in the play axis, press Play, and watch how the bubbles change size and position over the time span in your dataset, visualizing trends and relationships in the data that would've been difficult to recognize in a simple pivot table.

Now, let's say your field office has a call center that handles complaints and requests over the phone. Your requirements dictate a report that shows the satisfaction levels for each call center operator. With the Power View scatter diagram, you can put the satisfaction, urgency, and call center operator columns in the axes on the chart,

and the time column on the play axis. Then, the users of this report can watch how the satisfaction increases or decreases over time for each user as the urgency of the cases fluctuates, as shown in **Figure 3**.

Or, suppose in the past you would've created a pivot table and chart that break down the data by geography, giving you collapsible row groupings and bars that contain data for different geographical areas. But if you have a lot of these areas, understanding what's different about the data in the areas doesn't exactly jump out at you. You may do better by using Power Map, which can plot the values out graphically on a map control to help you identify where your values are large, where they're small, and what trends might exist geographically.

For example, a well-grouped pivot table or chart might show you that values in the West are generally higher than those in the East, but it wouldn't necessarily show you that values in coastal areas are much higher than those in inland areas. Power Map makes it much more intuitive and efficient to explore, identify, analyze and share insights about trends in data including a geographic dimension. **Figure 4** shows an example.

Finally, let's suppose you need to supplement the data from your application with other publicly available data. Specifically, you need to compare how the satisfaction of customers is related to whether their county voted Republican or Democrat in the last election. You find the voting data online, and you use Power Query to pull it into Excel. You use Power Pivot to create the relationships between the voting data and the data from your government business application. The voting data is now part of your data model, and you can include it in your data analysis and visualization in the same manner in which you've reported on the data that came from your application.

The integration with Excel and the self-service BI tools we've described allows you to put BI in the hands of your users more easily than ever before.

Enterprise Scalability

Your application is going to start with a small pilot, but within a few years, it will host thousands of users. If you build your application on Dynamics CRM, you'll benefit from its highly scalable architecture, which enables installation in a wide range of infrastructure configurations. The scalability of Dynamics CRM is a topic worthy of its own article, but you can get an overview by reviewing the TechNet Library white paper, "Microsoft Dynamics CRM 2011 Performance and Scalability on Intel Processor-Based Servers with Solid-State Drives," at technet.microsoft.com/library/hh204514.

Whether you build your application on the version of Dynamics CRM that runs in your own on-premises environment or leverage the Microsoft Software as a Service (SaaS) offering called Dynamics CRM Online, the same building blocks are used to scale out the deployment, because Dynamics CRM Online runs the same codebase you install in your own datacenter. This brings up another enormous benefit of using Dynamics CRM to build your application: By following some simple design principles, you can ensure your application will be portable across on-premises *and* online deployments.

Document Management and Storage

Out of the box, Dynamics CRM integrates with SharePoint by enabling documents in SharePoint to be associated with relational data in Dynamics CRM. To accomplish this, Dynamics CRM stores metadata about SharePoint folders that contain documents related to records in Dynamics CRM.

Although it appears to end users that the documents are stored in your application, they're actually stored in SharePoint behind the scenes. This means you can take advantage of all the inherent document-management capabilities of SharePoint (version control, document templates and the like). However, many customers and partners implement much more advanced integrations of Dynamics CRM and SharePoint by leveraging the SDKs for the two products in concert. This opens up numerous integration possibilities, as the following examples indicate:

- Automation of document library creation and custom document library folder structures
- Attaching Dynamics CRM data as metadata to documents in SharePoint
- Advanced mapping of the Dynamics CRM security model to limit access to SharePoint documents
- Surfacing Dynamics CRM data in SharePoint using Business Connectivity Services (BCS)
- Indexing and searching relational data in Dynamics CRM through SharePoint to provide a more Bing-like search facility

Granular, Role-Based Security

Your users aren't all the same. They perform different tasks and have different responsibilities, and in a government organization where

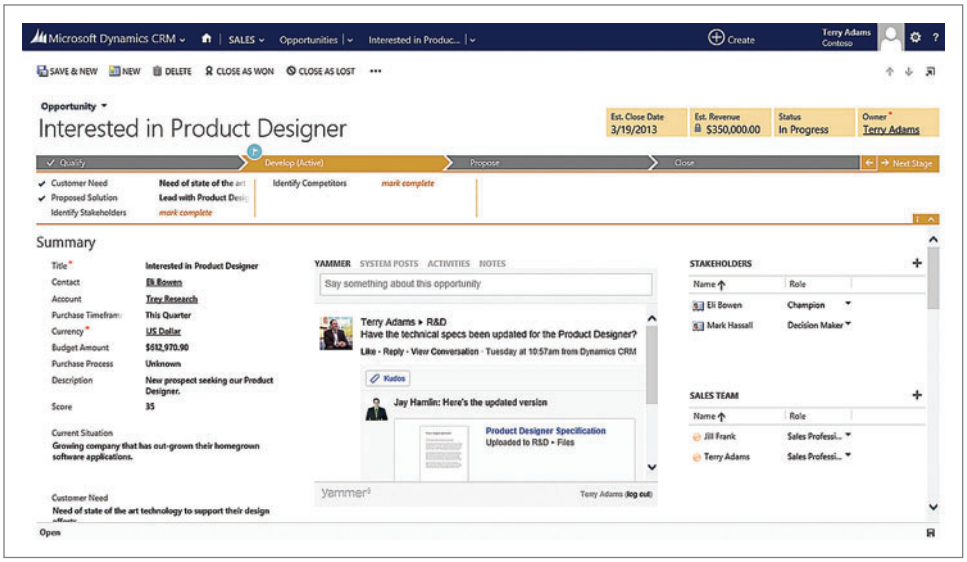


Figure 5 **A Touch-Friendly Web App**

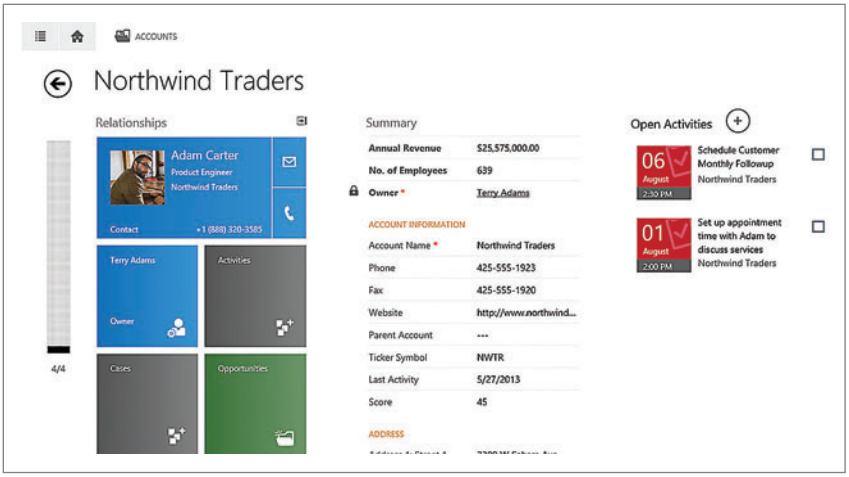


Figure 6 **A Touch-Optimized Native App**

security matters, it's important to ensure each user can access only the information required for his job. If you build your application from scratch, you'll need to figure out how to handle this security requirement, including how to make it easy to administer and how to address various exceptions that might arise in the course of business.

If you build your application on Dynamics CRM, you start your project with a granular security model. Your application will also have tools to handle security exceptions, such as record sharing, teams and team ownership, and team-selling features, all of which enable your users to design a security model that can meet stringent demands. For scenarios that require even more unique security rules, your application can use the Dynamics CRM SDK to further control data access by your end users.

Mobility

Because the new modern Dynamics CRM UI is touch-friendly, it works as well in a desktop Web browser as it does in the built-in browsers for iPad, Android and Windows 8 devices. If you want to go from touch-friendly (**Figure 5**) to touch-optimized (**Figure 6**), however, there are two possibilities. The first is the mobile application that comes with Dynamics CRM 2013. The second is building a custom mobile app.

Native Mobile App The trend in many industries toward a mobile workforce gave Microsoft the opportunity to develop a modern, streamlined application customized for the tablet experience. This application is available for Windows 8 and iPad devices, and it contains a set of high-value capabilities that cover the most likely tasks to be performed in a mobile scenario. For those building government business applications with a mobile component, this is another powerful platform benefit that speeds time-to-value by reducing the amount of work required by developers. In many cases, this new native mobile app eliminates the need to write a custom tablet app for XRM solutions. However, there will still be cases in which building a custom app will be a more appropriate solution.

Custom Mobile App If you're reading this article, you probably already know about the popularity of REST and OData with mobile developers and about the importance of OAuth in securing RESTful APIs, including OData. While Dynamics CRM 2011 exposed a RESTful OData endpoint for create, read, update and delete (CRUD) operations on CRM entities, it didn't have a supported mechanism for allowing external callers to interact with the OData service endpoint. Dynamics CRM 2013 brings full support for OAuth through Windows Azure Active Directory (WAAD) in Dynamics CRM Online and through the version of Active Directory Federation Services (AD FS) that ships with Windows Server 2012 R2. As a result, an OData endpoint is now available to external callers, exposing basic CRUD operations.

Together, these technologies provide a much-improved interoperability story for Dynamics CRM 2013, a story that includes a greatly enhanced developer experience for writing custom Windows Store apps and Windows Phone applications that use Dynamics CRM as their application server.

Integration with Other Systems

You run a lot of legacy applications in your organization—that's just the way it is. While you'd love to see these applications make way for newer applications across the infrastructure, you know that many of them will live as long as a tortoise. The data in these applications is complementary to the application you're building, and you need to integrate the two systems.

Built on Dynamics CRM, your application has a fanny-pack full of integration capabilities. These begin with the Web services and the SDK and end with third-party tools that allow you to set up integrations fairly easily. Your choice in executing this requirement depends on the nature of the legacy applications and the type of data in the legacy systems.

Wrapping Up

Our intention in this article was to demonstrate the many reasons why you should consider Dynamics CRM when evaluating technologies for building a government business application. It offers a powerful platform for building a wide range of apps, and offers many benefits for developers that result in a shortened time-to-value. With the release of Dynamics CRM 2013, applications will benefit even more from an improved modern UX, tablet mobility and new platform capabilities. ■

Marc Schweigert *is the chief architect of the U.S. Public Sector Dynamics team at Microsoft, where he has technical oversight of the pre- and post-sales solutions that Microsoft federal, state and local government, education, and health-care customers and partners are building using Dynamics CRM.*

Andy Schultz *loves reading, writing, learning, getting better at things, and participating in the advancement of the capabilities of humanity through technology. He's a husband, the dad of four, a BYU alum, and a technical architect focused on the public sector at Microsoft.*

Thanks to the following Microsoft technical experts for reviewing this article: *Girish Raja and Jim Toland*

GdPicture.NET 10

- Document viewing, processing, printing and scanning (TWAIN & WIA).
- Reading, writing and converting vector and raster images in more than 90 formats, PDF included.
- OMR, OCR, barcode reading and writing (linear & 2D).
- Annotations for image and PDF within Windows and Web applications.
- Color detection engine for image and PDF compression.

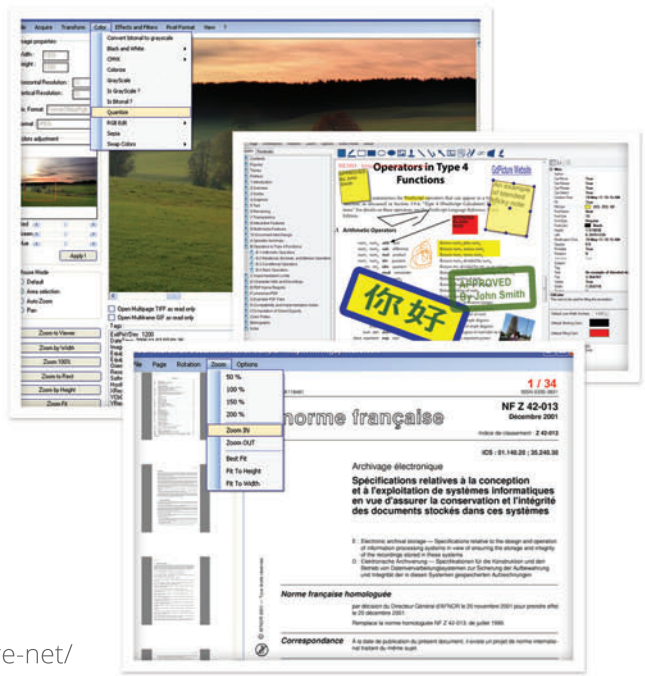
And much more ...

All-In-One Document Imaging SDK

Royalty-Free Document Imaging Toolkits for .NET and COM/ActiveX

GdPicture.NET 10 Plugins

- NEW Color detection
- Full managed PDF support
- Full annotations support for PDF and images
- OCR
- Forms processing
- JBIG2 encoding
- 1D and 2D barcode reading and writing



Try GdPicture.NET 10 FREE for 30 days

www.componentsource.com/products/gdpicture-net/

Singing Your Song

My youngest daughter came home from gymnastics class the other day, bubbling with enthusiasm. "Daddy, I won my team's unsung hero award!" she exclaimed. "That means I'm awesome, but nobody ever says it."

That describes you, my friends, does it not? Working for (or with) the government is rarely glorious. NASA moon landing moments are few and far between. You work hard and do your best, but does anyone even once say thank you? Does anyone even notice you're alive, except when a headline blares "Computer error exposes a million Social Security numbers"?

But where would the country be without your efforts? What if retirees' Social Security checks didn't arrive? What if the CDC couldn't track next winter's flu? Imagine if jet fuel or bullets—or even coffee—didn't get ordered for the next military expedition? The entire country would screech to a halt.

So I hereby acknowledge your efforts and your place in making our country work. And I do thank you, from the bottom of my heart.

You caught me in a really good mood. Yesterday I renewed my driver's license online, without schlepping to the registry. That sounds tiny, but multiply it by a million drivers per week, and someone did a very good thing by putting government services on the Web.

You're already doing a good job, and you must be reading this magazine in hopes of doing an even better job. Good for you. Regardless of any budget crunch, that's within your reach. It just takes careful thought. You can make any program better by improving its UX.

Consider how Google has evolved its user interaction over time. At first it was a Web page you had to call up. Then it was a toolbar always present in your browser. Then it started suggesting topics as you typed. Today it even pre-fetches pages as you type. Google is ruthlessly, even brutally, driving down the effort required to use its service. A millisecond here, two milliseconds there, times a billion queries per day ... it adds up fast.

Google accomplishes all of this without training its users. On the contrary, the company is constantly refining its UX to match the user thought process ("I'm feeling lucky"), not the other way around. Google uses computers for what computers are good at, so humans can do what humans are good at. You should aim for the same.

Know Thy User

Google religiously follows my First, Last and Only Law of UX Design: "Know Thy User, for He Is Not Thee." Like a Dale Carnegie rule, it's extremely simple but demands conscious effort to actually apply. You have to practice it every day until it becomes part of you. As my daughter likes to say, "Never criticize a man until you've walked a mile in his shoes. Then when you do start criticizing him, you have a mile head start and he has to chase you in his socks."

Read the classics on UX: "The Design of Everyday Things" by Donald Norman (Basic Books, 2002), "The Inmates Are Running the Asylum" by Alan Cooper (Sams-Pearson Education, 2004) and my own humble offering, "Why Software Sucks" (Addison-Wesley Professional, 2006). Take a class in UX design, like the ones I teach (bit.ly/16fuC03). Aim to make software just work.

Google uses computers for what computers are good at, so humans can do what humans are good at. You should aim for the same.

That will perfectly position you for your next huge project where you'll be putting your public data where anyone with an idea can use it. Read "Big Data" (Eamon Dolan/Houghton Mifflin Harcourt, 2013) by Viktor Mayer-Schonberger and Kenneth Cukier. Learn how Google managed to track the flu ahead of the CDC by mining its search term usage. Someone out there can do something you never even imagined with the public data you're holding, once you open it up to them. And you have the key.

When you and I meet someday, I'd really like to hear how you took my simple law and applied it to your program. If you liked this article, I hope you'll read my regular column in *MSDN Magazine*. You can find plenty of my work at bit.ly/no9xLK. Think of it as a cross between Andy Rooney, Dennis Miller and George Will. I'll warn you right now, though, I'm not always this nice. ■

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.*