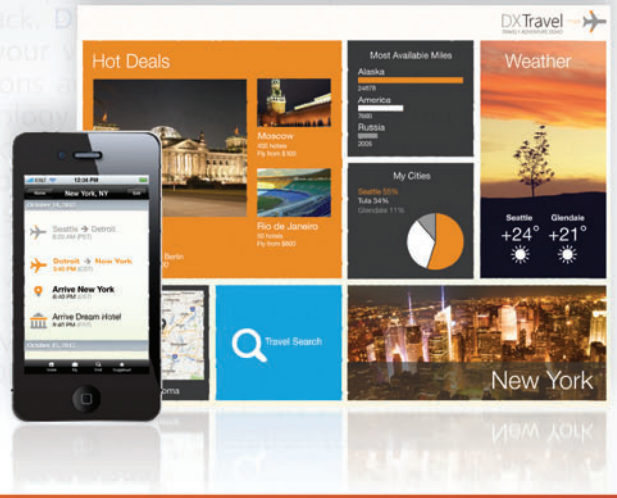
DevExpress

Delight your users by creating apps that feel as though they were designed expressly for the device. With **DXTREME**, multi-channel means building applications that span devices and optimize the best parts of each platform. And with HTML5/JS visualization built inv your dynamic charts and graphs will be both powerful and beautiful.



discover DXTREME

Are you ready to go cross-platform? Download the **DXTREME** Preview to experience the future.

www.DevExpress.com

DXv2

The next generation of inspiring tools. **Today.**

u | dxp | dxt | asp | win | wpf | sl | 8

THE MICROSOFT JOURNAL FOR DEVELOPERS

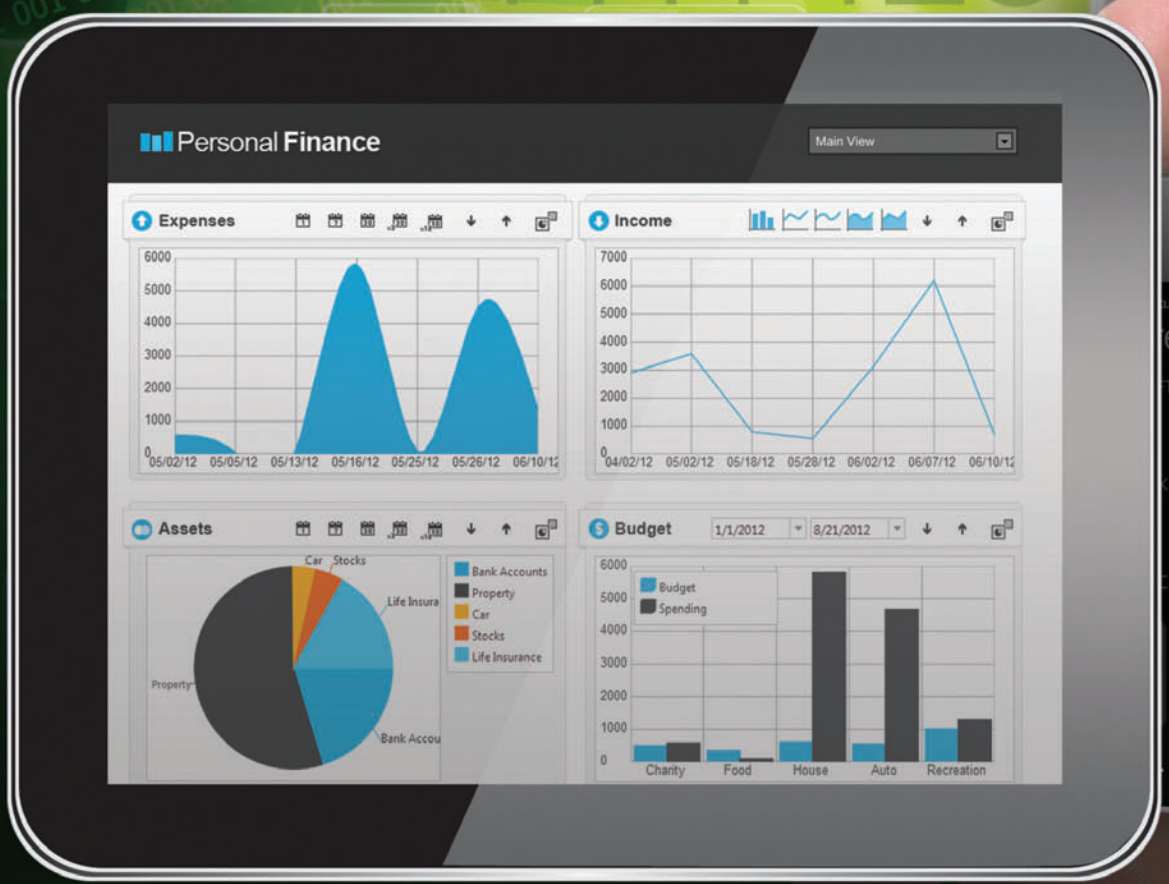
DECEMBER 2012 VOL 27 NO 12

msdn magazine

COLUMNS

Microsoft

Compatible with
Microsoft® Visual Studio® 2012



BRILLIANT UX

At Your Fingertips

infragistics.com/ **EXPERIENCE**

INFRAGISTICS™

DESIGN / DEVELOP / EXPERIENCE

Infragistics Sales US 800 231 8588 • Europe +44 (0) 800 298 9055 • India +91 80 4151 8042 • APAC (+61) 3 9982 4545

msdn magazine

DECEMBER 2012 VOLUME 27 NUMBER 12

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

LEADTOOLS®
WinRT SDKs

DEVELOP WINDOWS STORE APPLICATIONS IN C++/CX, C#, VB, JAVASCRIPT

LOAD, SAVE, PROCESS AND DISPLAY OVER 150 IMAGE FORMATS

VIEWER CONTROLS WITH MULTI-TOUCH SUPPORT ANNOTATIONS AND MARKUP

MULTI LANGUAGE OCR, BARCODE & PDF

DICOM DATA SET AND PACS

DOWNLOAD OUR 60 DAY EVALUATION
WWW.LEADTOOLS.COM

LEAD TECHNOLOGIES INCORPORATED

SALES@LEADTOOLS.COM
800.637.1840

MICHAEL DESMOND

Welcome Windows Azure Insider

In the Windows 8 special edition of *MSDN Magazine* in October, we introduced a new monthly column to our magazine readers. Windows Azure Insider offers a reinvigorated look at the rapidly evolving Microsoft cloud platform, and provides focused and timely hands-on guidance for developers working with Windows Azure.

In the Windows 8 special edition, authors Bruno Terkaly and Ricardo Villalobos showed how a Windows Store app can consume JSON data returned from a Web service deployed to the cloud via Windows Azure ("Windows 8 and Windows Azure: Convergence in the Cloud," msdn.microsoft.com/magazine/jj660302). In the November issue, Terkaly and Villalobos dove into Windows Azure Mobile Services (WAMS) and how it can be used to simplify implementation of software architectures that must address multiple device/OS types, rely on async Web services and manage unpredictable traffic flows ("Windows Azure Mobile Services: A Simple, Scalable and Robust Back End for Your Device Applications," msdn.microsoft.com/magazine/jj721590).

In this issue, the column plumbs the Windows Azure Service Bus. Terkaly and Villalobos show how the publisher/subscriber messaging pattern can be used to control the way messages are distributed based on rules and filters ("Windows Azure Service Bus: Messaging Patterns Using Sessions," p. 22).

Meet the New Bosses

If some of this sounds familiar, it could be you ran across Terkaly and Villalobos' work for us over the past year. In our February 2012 issue, the two wrote the feature titled "Building a Massively Scalable Platform for Consumer Devices on Windows Azure" (msdn.microsoft.com/magazine/hh781021), and in June they wrote "Democratizing Video Content with Windows Azure Media Services" (msdn.microsoft.com/magazine/jj133821). They've also been writing the Windows Azure Insider column for our Web site since March, which they kicked off with a column titled "Write Scalable, Server-Side JavaScript Applications with Node.js" (msdn.microsoft.com/magazine/hh875173).

So who are these guys? Both Terkaly and Villalobos are Microsoft evangelists who are immersed in Windows Azure development. While both share a passion for working with developers to help overcome challenges, they come from very different backgrounds.

Villalobos is a lifetime developer who has been in the programming business for nearly three decades. After getting a degree in electronics and communications technology, Villalobos worked in supply chain management and logistics. He later went back to get his MBA in order to, as he says, "better understand the logic and procedures behind the industry I was working in."

Terkaly got his start in finance and accounting, wrestling Lotus 1-2-3 spreadsheets for a large accounting firm. A desire to automate tasks led him to macro programming, which in turn made him curious about software development. As he says: "I ended up quitting the accounting job and began working as a C programmer for a public utility." He later worked for years as a Microsoft field engineer, a role he describes as "deeply technical."

At Microsoft, Terkaly and Villalobos are in a unique position to understand and address the technical challenges facing developers working with the Microsoft cloud platform.

Says Villalobos: "When Microsoft offered me the opportunity to become a Windows Azure cloud evangelist two years ago, it felt like all the efforts and learning of my life had found a perfect place to be applied: creating solutions, architecting distributed environments, mentoring developers through articles and presentations, and designing new business models with company executives."

We'll be learning a lot from Terkaly and Villalobos in the months to come, but their advice for now is simple. Terkaly urges developers to avoid thinking of cloud adoption as an all-or-nothing deal. Dev shops can attack database and Web-app migration in stages, before moving on to things like content distribution and identity management. He also says developers don't need to think they're starting over.

"In a general sense, as long as you think of your applications as being stateless, you can use the same languages, IDEs and tools as in a non-cloud world," he says.

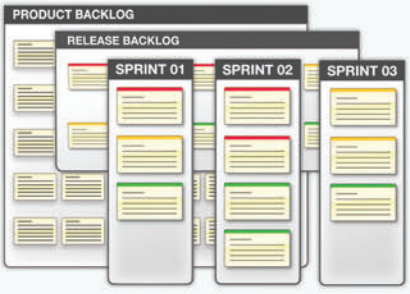
Villalobos agrees. "Understand that there are no black boxes in Windows Azure," he says. "At the end of the day, it's about working with servers and operating systems that they already know and understand."

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

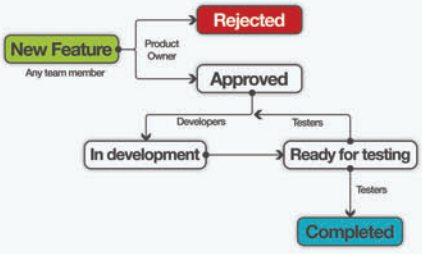
OnTime Scrum

Agile project management & bug tracking software

The **Scrum project management** tool your development team will **love to use.**



Easily manage product backlogs



Automate your workflow process

Project visibility with burndowns

Enterprise-quality software at prices **any team can afford.**

$10 per month for up to 10 users

special small-team pricing

$7 per user per month

for teams of 11+ users

Last chance! Exclusive offer for MSDN readers: **3 months free** of **OnTime Scrum**

Visit **OnTimeNow.com/MSDN** • **Expires 12/31/2012**

OnTime Scrum offers your dev team:

- flexible product backlog management
- powerful user story & bug tracking
- highly configurable user roles & workflows
- automated burndown charts
- **new** real-time visual project dashboard
- Visual Studio & Github integration
- all in a fast, versatile HTML5 interface

axosoft

800.653.0024 • www.ontimenow.com • www.axosoft.com • @axosoft

Essential Facebook Programming: Authentication and Updates

Facebook is a rich and complex software platform, which exposes a sophisticated and multifaceted development framework to developers. At first, the meaning of "Facebook programming" might not be clear. There are essentially two types of apps related to Facebook. One type comprises apps that live and thrive within the Facebook environment. These apps are essentially rich Web pages loaded in Facebook canvas pages and hosted in the main site. To use the app, users need to navigate to the Facebook site and log in to their own account. These apps can implement their own logic—whatever you can express from within a Web page using JavaScript or other Web programming technologies—and can gain access to Facebook goodies such as friends, news feeds, media and more. To get started on this form of programming, just go to the "Apps on Facebook.com" page at bit.ly/f5hERV.

Another approach to Facebook programming involves integrating some core Facebook functionalities into existing apps, such as Web sites, mobile apps (for example, Android, iOS or Windows Phone) or desktop applications.

In this article I'll focus on this aspect of Facebook programming and discuss how to use the Facebook C# API to authenticate users and post programmatically on behalf of the currently logged-in user.

Embedding the Like Button

When it comes to popular social networks such as Facebook and Twitter, the first level of integration with external apps is through the use of ad hoc buttons to "like" the page or tweet about it. Hardly any programming is required; it's purely a matter of inserting some ad hoc markup into the Web pages.

So the simplest way to make your Web site more popular is to embed the Facebook Like button in a new iframe so that any user visiting the page can immediately like it on Facebook. Here's the minimal markup you need:

```
<iframe src="http://www.facebook.com/plugins/like.php?href=XXX">
</iframe>
```

You replace XXX with the URL of the page to like. In addition, you might want to add a bit of CSS style to the iframe element to make it better merge with the rest of the page. **Figure 1** shows the final result.

The Like button is the simplest (and most popular) of the Facebook social plug-ins. Most of the time, you can integrate plug-ins in your Web pages via frames or ad hoc markup. Some plug-ins require the Facebook JavaScript SDK, and some only work if you have custom Facebook pages. I'll discuss scripting Facebook in a future column.

Beyond using social plug-ins, integrating Facebook in apps (and not just Web sites) means being able to perform two main tasks: let users authenticate themselves with your app using their Facebook account, and enable the app to post to the Facebook walls of particular users. Let's start with user authentication.

> The Like button is the simplest (and most popular) of the Facebook social plug-ins.

OAuth and the Old Dream of a Single Authentication Module

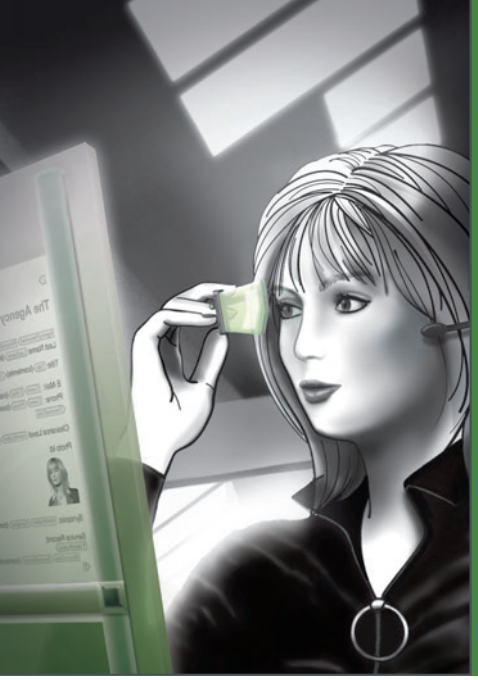
User authentication is a core function of just about any significant Web site. A decade ago, one of the best-selling points of ASP.NET in comparison to classic ASP was the availability of a highly reusable membership system that facilitated the development of an authentication layer in a fraction of the time normally required.

These days, however, having a custom authentication layer is an enticing option. By implementing an ad hoc authentication layer, developers make themselves responsible for safely storing passwords and for the costs of managing thousands of accounts or more. For users, that means yet another username/password pair to remember.

Years ago, the Microsoft Passport initiative was a smart but probably too-early attempt to make users' lives easier when they moved across a few related sites. The idea behind Passport was that users just needed a single successful logon to freely navigate through all of the associated sites.

Figure 1 **The Facebook Like Button**

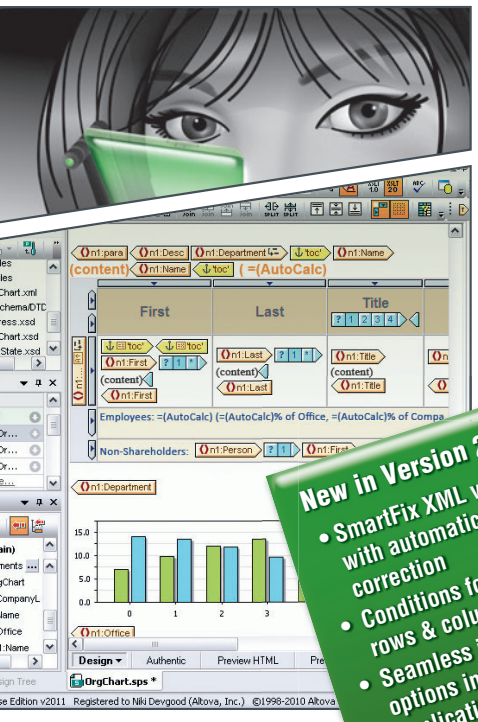
Code download available at archive.msdn.microsoft.com/mag201212CuttingEdge.



Uncover Web development simplicity with the complete set of tools from Altova®

ALTOVA® missionkit®

Experience how Altova MissionKit®, the integrated suite of XML, SQL, and UML tools, can simplify even the most advanced Web 2.0 development projects.



Altova MissionKit includes powerful Web development tools:

XMLSpy® – advanced XML editor
- Intelligent HTML5 & CSS3 editing
- XSLT 1.0/2.0 editing help, plus debugging & profiling

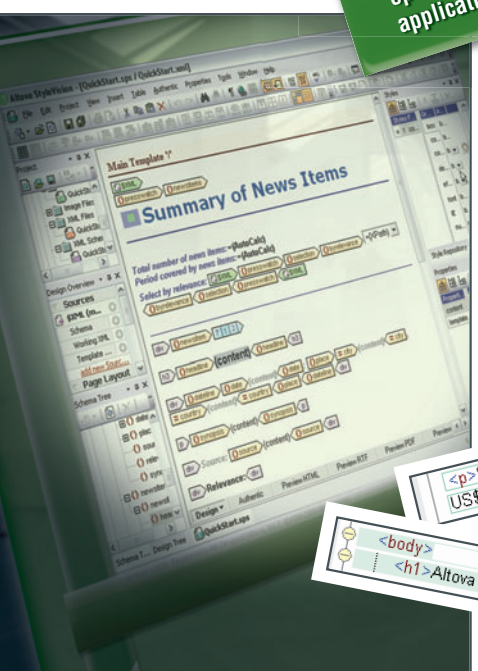
StyleVision® – graphical stylesheet & report designer
- Drag-and-drop XSLT 1.0/2.0 stylesheet and Web page design
- Powerful XML, XBRL, & database report builder
- Advanced CSS and JavaScript functionality

DiffDog® – XML-aware diff / merge utility
- File, folder, directory, DB comparison and merging
- One-click directory syncing; Web and FTP server support

New in Version 2013:
- SmartFix XML validation with automatic error correction
- Conditions for table rows & columns
- Seamless integration options in Java applications

Download a 30 day free trial!

Try before you buy with a free, fully functional trial from **www.altova.com**.



Scan to learn more about Web development with MissionKit.

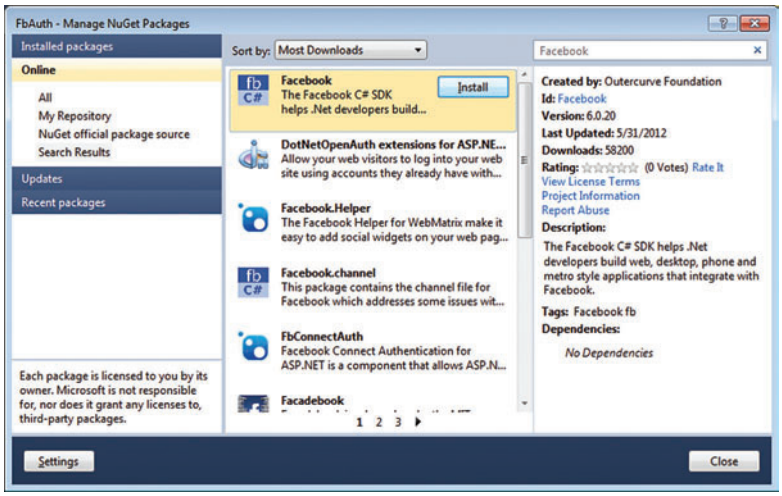


Figure 2 **Referencing the Facebook C# SDK via NuGet**

Bundled with older versions of ASP.NET, the Passport API is now officially gone. The problem it was devised to solve, however, is still present.

Today, OpenID (openid.net) is a great option for Web sites that need to authenticate their users but don't want to charge them with yet another set of credentials. OpenID is a single sign-on (SSO) protocol that your site uses to connect to a third-party service provider that will manage the authentication for you. An OpenID-enabled site manages only the beginning and end of the authentication task. It redirects the user to the configured OpenID provider and gets identity information back when everything has happened.

While OpenID is a popular buzzword, another one is being discussed even more: OAuth (oauth.net). What's the difference?

OpenID is exclusively an SSO protocol. An OpenID provider only manages the identities of registered users. But social networks such as Twitter and Facebook need more. One app might simply want to let users authenticate via Facebook. Another might want to authenticate via Facebook but also post to the user's wall. Yet another app might want to read the user's timeline and recent activity. On top of everything, there's authentication (which can be managed via the OpenID protocol), but the same user may grant different permissions to different apps. And this is an aspect that OpenID was not designed to handle.

OAuth is the protocol that Twitter and Facebook use to handle authentication and authorization. An OAuth provider doesn't simply return identity information. It asks the user which permissions he wants to grant to the app and then packages everything, credentials and permissions, into an access token. The client app will then pass the access token to perform any permitted operation on behalf of the user. One benefit of OAuth (and OpenID, as well) is that the provider never discloses user credentials to the app. In addition, when OAuth is used, the end user can revoke permissions to any app at any time. So, for example, imagine that at some point you authenticate with app XYZ using Twitter (or Facebook) and grant XYZ permission to post on your behalf. You can revoke this permission at any time by simply going to your Twitter (or Facebook) profile page. Note that the access token returned by the OAuth protocol is application- and user-specific. The user will need to log in to multiple apps if he intends to operate on multiple OAuth-based apps. OAuth is an HTTP-based protocol for authenticating users. You don't often have to write HTTP requests manually, even though you could (for example, to learn how things work "under the covers"). In the Microsoft .NET Framework, you can use general-purpose libraries such as DotNetOpenAuth (dotnetopenauth.net) or pick up ready-made frameworks for a specific social network such as TweetSharp for Twitter and the Facebook C# SDK for Facebook.

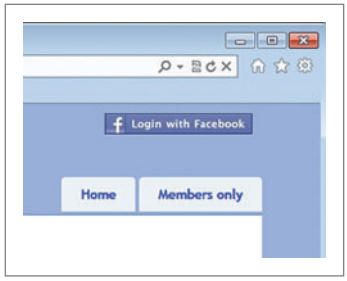


Figure 3 **The Button to Trigger the Facebook Authentication Process**

When it comes to authentication, take a look at the Windows Azure Access Control Service. This can act as a federated identity provider and can translate multiple identity provider mappings. It can host OAuth and Security Assertion Markup Language-based tokens from Active Directory. It's also built to work with Facebook and the new ClaimsPrincipal class in the .NET Framework.

Authenticating Users via Facebook

Let's start with a basic ASP.NET MVC site and use NuGet to plug in the Facebook C# SDK (see **Figure 2**). The ASP.NET MVC sample site is enriched with a brand-new authentication controller with three methods, which I named FacebookLogin, FacebookAuthenticated and Logoff. The first two methods represent the two steps of an OAuth interaction; the Logoff method just logs the user out of the host app. Note that the Logoff action isn't supposed to log the user out of Facebook. OAuth and the Facebook C# SDK only manage the authentication logic—persisting authentication information via cookies is still up to the ASP.NET MVC site.

The homepage of the site provides a link for the user to click when she wants to log in (see **Figure 3**). The link can simply be an anchor to an image (or a text) that points to the FacebookLogin action on the authentication controller:

```
<a href="/Auth/Logon" >
  <img src="@Url.Content("~/Content/Images/loginfb.png")"
    style="border:0" alt="Sign in with FB" />
</a>
```

In a standard ASP.NET MVC site, this markup belongs to the _logOnPartial.cshtml page. Let's have a look at the authentication controller.

In the FacebookLogin action—the name is arbitrary—you need to do a couple of things. First, you arrange the URL that, once invoked from the OAuth provider, will cause your app to complete the authentication step. In the sample app I'm considering, the URL will point to the FacebookAuthenticated action on the Auth controller. You can use the handy UriBuilder class for this kind of work:

Review Assistant

Code review is dull and tedious?

Make it fast and easy!

Review Assistant

Code review add-in for Visual Studio.

Integrates with:
TFS, SVN, Mercurial, Git

devart
WWW.DEVART.COM

Figure 4 **Login Parameters**

Parameter	Description
client_id	ID of the Facebook app acting as the proxy between the user app and the Facebook site. You get this unique string when you register your Facebook app.
redirect_uri	URL to return to complete authentication (for example, grab identity information and create an authentication cookie).
response_type	This can be "token" or "code," and it refers to how Facebook returns the access token after a successful authentication. For Web sites, it should be "code," as this ensures the access token—a sensitive piece of information—is not appended to the URL.
Scope	Indicates the additional permissions you want to request for the user. When the scope parameter is blank, the app has access to the user's basic information such as name, picture and gender. Through this parameter, you can request the e-mail address, as well as the publishing stream to be able to post. The full list of permissions is available at bit.ly/NCcAgf.

```
var uri = new UriBuilder(Request.Url)
  {
    Path = Url.Action("FacebookAuthenticated", "Auth")
  };
```

The second thing to do in the FacebookLogin action is arrange the Facebook URL for login. If you use the Facebook C# SDK, here's the code you need:

```
var client = new FacebookClient();
var appId = ConfigurationManager.AppSettings["fb_key"];
var returnUri = uri.Uri.AbsoluteUri;
var fbLoginUri = client.GetLoginUrl(new
  {
    client_id = appId,
    redirect_uri = returnUri,
    response_type = "code",
    scope = "email"
  });
```

The parameters you pass to the GetLoginUrl method help prepare the URL. **Figure 4** describes the parameters in more detail. The action FacebookLogin ends by redirecting the user to the returned login URL.

To authenticate a user via Facebook (or Twitter), you need to register a Facebook (or Twitter) app first and get some unique codes for that function. To register a new Facebook app, go to bit.ly/mRw8BK. Upon successful creation of the app, Facebook gives you an app ID string and an app secret string displayed in the page shown in **Figure 5**.

Both the app ID and app secret are required to perform Facebook operations from within a Web site or any other type of user app. It's common to store the app ID and app secret in the configuration file of the wrapper app:

```
<appSettings>
  <add key="fb_key" value="xxxxxxxxxxx"/>
  <add key="fb_secret" value="yyyyyyyyyyyyyyyyyyyy"/>
</appSettings>
```

The Facebook login URL does what it can to authenticate the user. In particular, if the user is currently logged in to Facebook, a request token is prepared and sent immediately to the specified return URL—in this case, this is the FacebookAuthenticated action. If no user is currently logged in on the machine, Facebook displays a classic login page. In doing so, it also lists the permissions that the app is requesting. At the minimum, permissions concern e-mail address and display name. However, they might include permission to post on behalf of the user or access to the media stream, friends, timeline and more. The user is expressly requested to approve or deny those permissions. Full permissions reference information can be found at bit.ly/P86tTC.

Finalizing the Authentication Process

To finalize authentication, you need to request and parse the access token, as shown in **Figure 6**. The method ParseOAuthCallbackUrl allows you to retrieve the access code from Facebook. This code isn't sufficient to control the Facebook account; you need to exchange it for an access token. According to the OAuth protocol, this requires another step and another HTTP request. Once your Web site holds the access token for a given user and a given Facebook app, it gains control of the Facebook account for the permissions the user explicitly granted. At the very minimum, you have permission to retrieve information about the user such as first and last name and, if allowed, e-mail address.

If your only purpose is authenticating users via Facebook, you create the standard ASP.NET authentication cookie and you're done. If your app wants to do more (for example, post on behalf of the user), then you need to store the access token so it can be given the same lifetime as the authentication cookie. You can save the access token to a database or to a cookie. Better yet, you can create a custom principal and store the access token as extra user data in the ASP.NET authentication cookie. In my sample app, I'm just creating an additional cookie that's managed by the FbHelpers class (see the accompanying source code for details).

Note, though, that the access token is subject to some expiration rules. If you authenticate

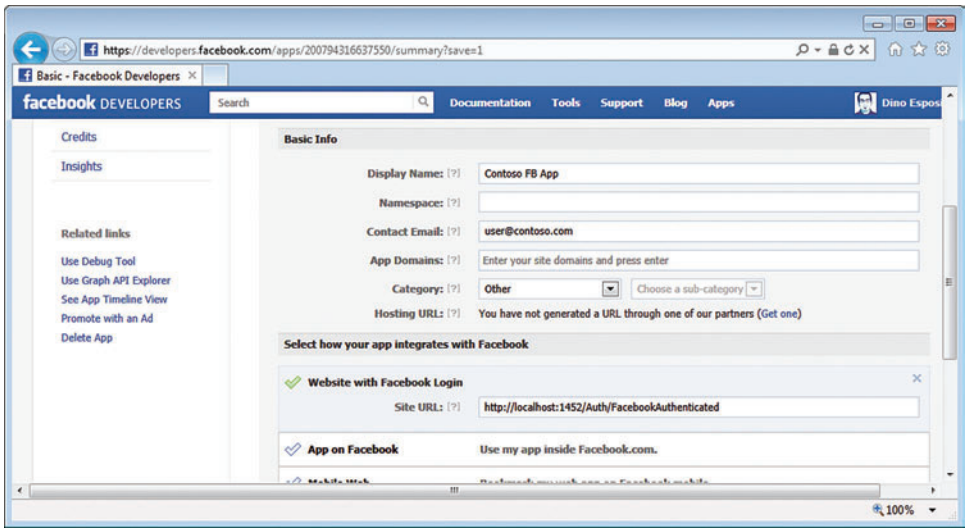


Figure 5 **Registering a New Facebook App**

The #1 Native Toolset for Building Windows 8 Apps

Build for the consumer and enterprise markets with either XAML or HTML

XAML HTML

www.telerik.com/win8

telerik
deliver more than expected

Figure 6 **Finalizing the Authentication**

```
public ActionResult FacebookAuthenticated(String returnUrl)
{
  // Prepare the return URL
  var returnUri = new UriBuilder(Request.Url) {
    Path = Url.Action("FacebookAuthenticated", "Auth")
  };

  // Parse response to get the access code
  var client = new FacebookClient();
  var oauthResult = client.ParseOAuthCallbackUrl(Request.Url);

  // Exchange the code for the access token
  dynamic result = client.Get("/oauth/access_token",
    new {
      client_id = ConfigurationManager.AppSettings["fb_key"],
      client_secret = ConfigurationManager.AppSettings["fb_secret"],
      redirect_uri = returnUri.Uri.AbsoluteUri,
      code = oauthResult.Code
  });

  // Saves the token to a cookie for further access
  var token = result.access_token;
  FbHelpers.AccessTokenSave(Response, token);

  // Grab identity information using the access token
  dynamic user = client.Get("/me",
    new {
      fields = "first_name,last_name,email",
      access_token = token
  });

  // Create the ASP.NET authentication cookie
  var userName = String.Format("{0} {1}",
    user.first_name, user.last_name);
  FormsAuthentication.SetAuthCookie(userName, false);

  // Back home
  return Redirect(returnUrl ?? "/");
}
```

using server-side code (that is, send users to authenticate with the Facebook site, as discussed here), then you get a long-lived token that lasts for 60 days. If you use the JavaScript SDK and attempt a client-side authentication, then you get a short-lived token that expires after two hours. You can extend this duration to 60 days by making a second call to a specific endpoint. In any case, once the access token is expired, users need to authenticate again and reacquire a valid access token. Here's a good way of coding that:

```
try {
  var client = new FacebookClient(...);
  dynamic result = client.Get("me/friends");
} catch (FacebookOAuthException) {
  // Your access token is invalid or expired.
}
```

The whole story is well-summarized at bit.ly/Qfeh5s.

Posting to a User's Wall

In an OAuth scenario, when you hold the access token for a user/application pair, you can programmatically execute any of the interactive operations for which the user granted permissions to the app. To post a message to the user's wall you need the following code:

```
public static void Post(String accessToken, String status)
{
  var client = new FacebookClient(accessToken);
  client.Post("/me/feed", new { message = status });
}
```

You call this helper method from within a controller action method. The access token string must be retrieved from wherever you saved it, whether a custom cookie, the authentication cookie or a persistent store. If you lost the access token, then for the post operation to be successful the user needs to log out and log in again. If you intend to perform tasks against Facebook, storing the access token in a way that survives browser restarts is key. **Figure 7** shows the sample app that posts a message, and the user's wall properly updated.

Next Up: Windows Presentation Foundation

To summarize, Facebook exposes a fairly rich API through which developers can integrate Facebook content and logic with their own apps. Facebook apps are certainly embedded apps living within the Facebook boundaries, but they're also classic Web or desktop applications living externally. In this column I used the Facebook C# SDK to perform two simple but quite common operations: authenticating a user of a Web site using her Facebook account and using the pages of the site to post to the current user's wall. In the next column, I'll expand the set of Facebook APIs used and discuss how to achieve the same operations from within a Windows Presentation Foundation application. ■

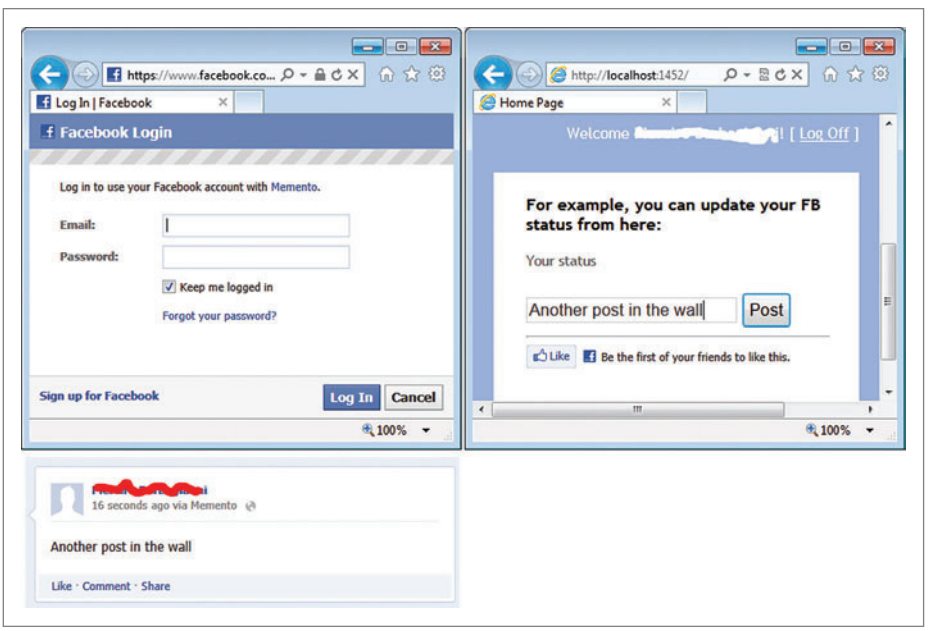


Figure 7 **Posting to the Wall**

Dino Esposito *is the author of "Architecting Mobile Solutions for the Enterprise" (Microsoft Press, 2012) and "Programming ASP.NET MVC 3" (Microsoft Press, 2011), and coauthor of "Microsoft .NET: Architecting Applications for the Enterprise" (Microsoft Press, 2008). Based in Italy, Esposito is a frequent speaker at industry events worldwide. Follow him on Twitter at twitter.com/despos.*

Thanks to the following technical expert for reviewing this article: *Scott Densmore*

JULIE LERMAN

Pain-Free Data Access in JavaScript—Yes, JavaScript

Although I think of myself as a plumber when it comes to software development, I have to do a fair amount of client-side development as well. Even in this column, I venture into client apps where they intersect with data access. But when the client side is JavaScript, it's never been pretty—my JavaScript skills, unfortunately, are still lacking and each learning curve is a world of pain for me (and for my Twitter followers, who are subjected to my venting). But it's always worth it when I push through the brick wall and get to success. And anything that makes working in JavaScript easier for me is something I welcome with open arms. So when, during a Vermont.NET User Group presentation on single-page applications, Ward Bell from IdeaBlade, gave us a look at an open source data access API for JavaScript that he and his team were cooking up, I was very interested. From the perspective of my experience with the Entity Framework, what I saw was comparable to using EF for client-side Web development. The API is called Breeze and at the time of this writing is in beta. Bell was generous with his time to help me learn more about Breeze for the sake of this column. You can get it at breezejs.com, where you'll also find an impressive array of documentation, videos and samples.

Figure 1 **Key Ingredients of My Web API Service**

```
readonly EFContextProvider<PersonModelContext> _contextProvider =
  new EFContextProvider<PersonModelContext>();

[AcceptVerbs("GET")]
public IQueryable<Person> People()
{
  return _contextProvider.Context.People;
}

[AcceptVerbs("GET")]
public IQueryable<Device> Devices()
{
  return _contextProvider.Context.Devices;
}
[AcceptVerbs("POST")]
public SaveResult SaveChanges(JObject saveBundle)
{
  return _contextProvider.SaveChanges(saveBundle);
}

[AcceptVerbs("GET")]
public string Metadata()
{
  return _contextProvider.Metadata();
}
```

This article discusses a beta version of Breeze. All information is subject to change.

Code download available at archive.msdn.microsoft.com/mag201212DataPoints.

My June 2012 Data Points column, "Data Bind OData in Web Apps with Knockout.js" (msdn.microsoft.com/magazine/jj133816), focused on using Knockout.js to more easily perform data binding on the client side. Breeze works seamlessly with Knockout, so I'm going to revisit the example from the June column. My goal is to see how introducing Breeze could simplify my coding effort in the example's workflow:

- Getting data from the server.
- Binding and presenting that data.
- Pushing changes back to the server.

> Anything that makes working in JavaScript easier for me is something I welcome with open arms.

I'll walk through the critical parts of an updated solution so you can see how the puzzle pieces fit together. If you want to follow along with a properly set-up solution and test things out, you can download the full solution from archive.msdn.microsoft.com/mag201212DataPoints.

The Original Sample

Here are the key steps of my earlier solution:

- On the client side, I defined a person class that Knockout can use for data binding:

```
function PersonViewModel(model) {
  model = model || {};
  var self = this;

  self.FirstName = ko.observable(model.Name || ' ');
  self.LastName = ko.observable(model.Name || ' ');

}
```

- My data was provided via an OData data service, so I accessed the data using datajs, a toolkit for consuming OData from JavaScript.
- I took the query results (which are returned as JSON) and created a PersonViewModel instance with the values.
- My app then let Knockout handle the data binding, which also coordinates changes made by the user.
- I took the modified PersonViewModel instance and updated my JSON object from its values.
- Finally, I passed the JSON object to datajs to save back to the server through OData.

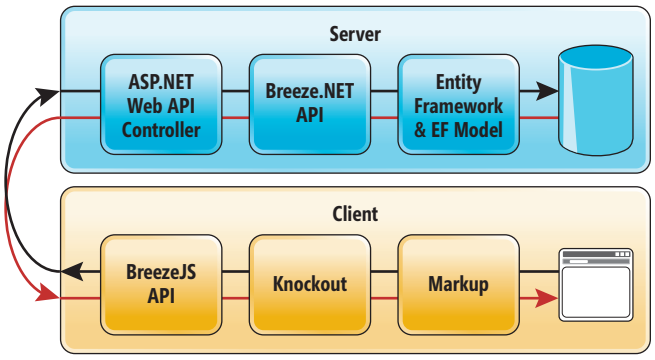


Figure 2 **The Breeze.NET API Helps on the Server While the BreezeJS API Helps on the Client**

I didn't even bother with related data, as it would have added a lot more complexity for that small sample.

An Updated Service Using ASP.NET Web API

With Breeze, I can make HTTP calls to my OData service or to a service defined by ASP.NET Web API (asp.net/web-api). I switched my service to ASP.NET Web API, which works against the same EF model I used previously—with one addition. My former sample exposed only Person data. I now have related data in the form of a Device class, as every developer I know has a small collection of personal devices. The relevant functions exposed by my ASP.NET Web API are a GET, which returns Person data; another GET for Device data; and a single POST for saving changes. I'm also using a Metadata function to expose the schema of my data, as shown in **Figure 1**. Breeze will use this Metadata in order to understand my model.

Breeze.NET on the Server

Take note of the _contextProvider variable used in these methods. I'm not calling methods of my EF DbContext (PersonModelContext) directly. Instead, I've wrapped it with the Breeze EFContextProvider. This is where the _contextProvider.Metadata method is coming from, as well as the signature of SaveChanges, which accepts a saveBundle parameter. With saveBundle, Breeze is going to let me send a set of data modifications from my app, which it will pass on to my DbContext to persist to the database.

I've named my ASP.NET Web API app "BreezyDevices," so now I can request schema using http://localhost:19428/api/breezydevices/metadata. And I can query for data by specifying one of the GET methods: http://localhost:19428/api/breezydevices/people.

Because Breeze on the client side will query and save to an ASP.NET Web API remote service, I can remove datajs from my client app.

How Breeze Will Help My Sample

In the scope of this sample, I'll use Breeze to focus on three pain points:

1. My service returns and accepts bare JSON, but I need to work with JavaScript objects with Knockout observable properties for data binding to the UI.
2. I want to include related data, but this is hard on the client.
3. I need to send multiple changes to the server for saving.

With Breeze, I can data bind directly to my resulting data. I'll configure Breeze to use Knockout and, in response, it will create Knockout observable properties for me under the covers. This means that working with related data is much simpler because I don't have to translate it from JSON to bindable objects, and I don't have to make the extra effort to redefine graphs on the client side using my query results.

There is some server-side configuration involved in using Breeze. I'll leave the details of that to the Breeze documentation so I can focus on the client-side data-access part of the sample. And because there's a lot more to Breeze than what I'll be leveraging in this sample, once I've given you the flavor of it you'll want to visit breezejs.com to learn more.

Figure 2 shows where Breeze fits into the workflow on the server side and on the client side.

Querying from Breeze

My experience with OData and with Entity Framework makes querying with Breeze familiar. I'll work with the Breeze EntityManager class. The EntityManager is able to read the data model supplied by your service's metadata and produce JavaScript "entity" objects on its own; you don't have to define entity classes or write mappers.

> With Breeze, I can data bind directly to my resulting data.

There's a bit of client-side configuration to do as well. For example, the following code snippet creates shortcuts to some Breeze namespaces and then configures Breeze to use Knockout and ASP.NET Web API:

```
var core = breeze.core,
          entityModel = breeze.entityModel;

core.config.setProperties({
  trackingImplementation: entityModel.entityTracking_ko,
  remoteAccessImplementation: entityModel.remoteAccess_webApi
});
```

It's possible to configure Breeze to use a number of alternative binding frameworks (such as Backbone.js or Windows Library for JavaScript) and data-access technologies (such as OData).

Next, I create an EntityManager that knows the relative uri of my service. The EntityManager is comparable to an Entity Framework or OData context. It acts as my gateway to Breeze and caches data:

```
var  manager = new entityModel.EntityManager('api/breezydevices');
```

Now I can define a query and have my EntityManager execute it for me. This code is not too dissimilar to using Entity Framework

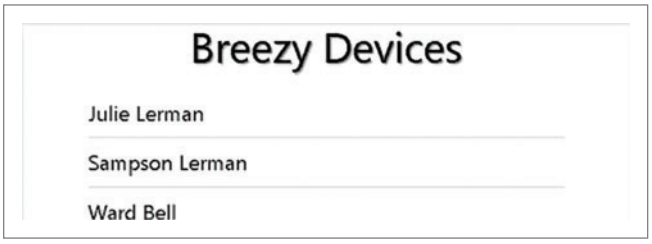
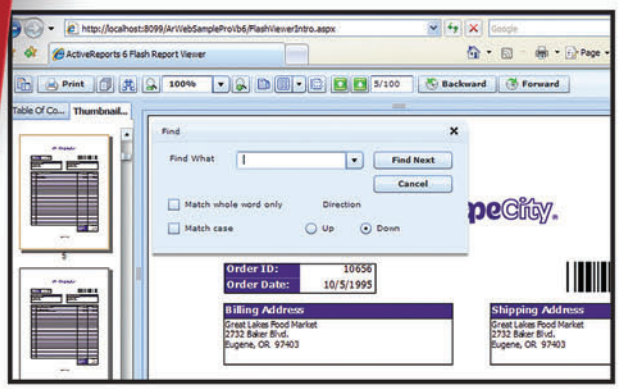
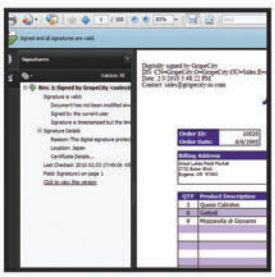


Figure 3 **Using Breeze and Knockout to Easily Consume Data in JavaScript**

<activereports>

platforms supported: windows forms · asp.net · silverlight · windows azure





ActiveReports 7
NEW VERSION

- Fast & flexible reporting engine
- Fixed page layout: create advanced reports
- Full report customization
- More controls: tables, matrices, charts, barcodes, lists & more

Follow the ActiveReports 7 road to success

It starts with a seamless Microsoft Visual Studio integration, advances with easy report creation and design from simple invoices and sales reports to complex tax forms and financial analysis reports, and continues with effortlessly exporting your .NET reports to popular formats. The road to success is wide open with endless customization options given the extensive APIs and integrated designers. Plus, maintenance is a cinch. Download your free trial and start delivering professional reports today.

FREE TRIAL @: http://c1.ms/reporting

"There's nothing like it right now in the market."

Dani Diaz
Developer Evangelist, Microsoft

EXPECT MORE.
GET MORE.

ComponentOne
a division of GrapeCity

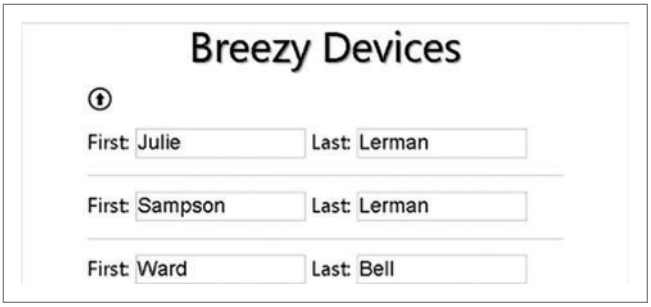


Figure 4 **Using Breeze to Save Data via JavaScript**

and LINQ to Entities, or to working with any of the OData client APIs, so it was my favorite part of learning how to use Breeze:

```
function getAllPersons(peopleArray) {
    var query = new entityModel.EntityQuery()
      .from("People")
      .orderBy("FirstName, LastName");

    return manager
      .executeQuery(query)
      .then(function (data) {
        processResults(data,peopleArray); })
      .fail(queryFailed);
  };
```

I'm doing this on the client side and can perform my query execution asynchronously, which is why the executeQuery method lets me define what to do when the query executes successfully (.then) as well as what to do if it fails (.fail).

> Expand is a term you might recognize from OData or NHibernate, and is similar to Include in the Entity Framework.

Notice that I'm passing an array (which you'll see shortly is a Knockout observable array) to getAllPersons. If the query execution succeeds, I pass that array on to the processResults method, which will empty the array and then populate it with the data from the service. Previously I would've had to iterate through the results and create each PersonViewModel instance myself. Using Breeze, I can use that returned data directly:

```
function processResults(data, peopleArray) {
    var persons = data.results;
    peopleArray.removeAll();
    persons.forEach(function (person) {
      peopleArray.push(person);
    });
  }
```

This gives me an array of person objects that I'll present in the view.

The getAllPersons function is inside of an object I've called dataservice. I'll use dataservice in the next bit of code.

A Self-Populating View Model

In the sample from my June Knockout article, the query and results were separate from the PersonViewModel class I used for the data binding in the view. So I executed the query and translated the results into a PersonViewModel instance with mapping code that I wrote. As I don't need mapping code or a PersonViewModel with Breeze, I'll make my app a bit smarter this time and have it display an array of Person objects fetched from the database by my dataservice. To reflect this, I now have an object named PeopleViewModel. This exposes a people property that I've defined as a Knockout observable array, which I populate using dataservice.getAllPersons:

```
(function (root) {
  var app = root.app;
  var dataservice = app.dataservice;

  var vm = {
    people: ko.observableArray([]),
    }
  };

  dataservice.getAllPersons(vm.people);
  app.peopleViewModel = vm;

}(window));
```

In the download sample you'll find a file called main.js, which is the starting point for the application logic. It contains the following line of code that calls the Knockout applyBindings method:

```
ko.applyBindings(app.peopleViewModel, $("content").get(0));
```

The applyBindings method connects the view model properties and methods to the HTML UI controls via the data bindings declared in the view.

The view in this case is a small chunk of HTML in my index.cshtml. Notice the Knockout data-bind markup that binds and displays the first and last name of each *person* object in the *people* array:

```
<ul data-bind="foreach: people">
  <li class="person" >
    <label data-bind="text: FirstName"></label>
    <label data-bind="text: LastName"></label>
  </li>
</ul>
```

When I run my app I get a read-only view of my person data, as shown in **Figure 3**.

Tweaking JavaScript and Knockout to Allow Editing

As you may recall from the June column, Knockout makes it easy to bind data for editing. Together with Breeze, this is a great combination for easily editing data and persisting it back to the server.

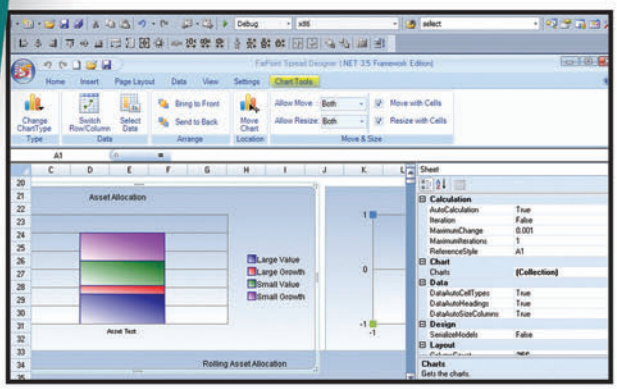
First, I add a function to the dataservice object that calls the Breeze manager.saveChanges method. When called, the Breeze EntityManager bundles up the pending changes and POSTs them to the Web API service:

```
function saveChanges() {
     manager.saveChanges();
  }
```

Figure 5 **Modifying the View to Display the Device Data**

```
<ul data-bind="foreach: people">
  <li class="person" >
    <form>
      <label>First: </label><input data-bind="value: FirstName" />
      <label>Last: </label> <input data-bind="value: LastName" />
      <br/>
      <label>Devices: </label>
      <ul class="device" data-bind="foreach: Devices">
        <li>
          <input data-bind="value: DeviceName"/>
        </li>
      </ul>
    </form>
  </li>
</ul>
```

<spread>

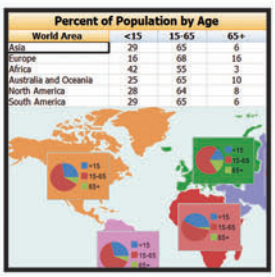


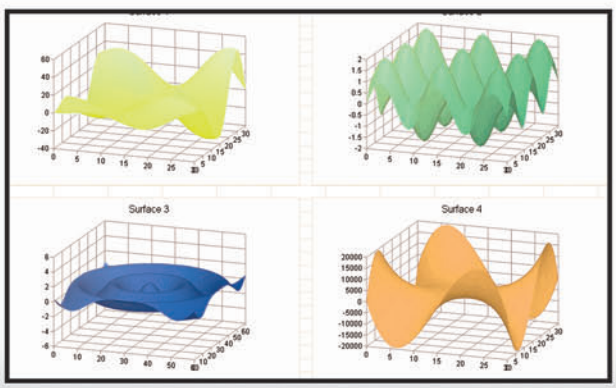
Spread's powerhouse components make it as easy as 1-2-3:

Import Microsoft Excel documents, preserving complete formatting

Interact with the data in Spread from within your application

Then export your spreadsheets to Microsoft Excel for portable distribution to your users





Where Easy & Excel Come Together in .NET – Spread.NET & Spread WPF-Silverlight

Ever been asked to add financial risk models, a budget analysis, or an engineering structural stress analysis to your application, and don't know how? With Spread.NET and Spread WPF-Silverlight you get the power, familiarity, and flexibility of Microsft Excel-compatible spreadsheets to harness in your business, engineering, and scientific applications. A vast Excel-functional library, data visualization, and enhanced file format support make these tasks effortless.

FREE TRIAL @: http://c1.ms/spreadsheets

"It's fast. It does what we need it to do. It just works."

Jeffrey Baron
VP App Software Development

EXPECT MORE.
GET MORE.

ComponentOne
a division of GrapeCity

Then I'll expose the new saveChanges function as a feature of the dataservice:

```
var dataservice = {
    getAllPersons: getAllPersons,
    saveChanges: saveChanges,
  };
```

Now my PeopleViewModel object needs to expose its own save method for binding to the view; the view model save function delegates to the dataservice saveChanges method. Here I use a JavaScript "anonymous function" to define the view model save:

```
var vm = {
    people: ko.observableArray([]),
    save: function () {
      dataservice.saveChanges();
    },
  };
```

Next, I replace my labels with input elements (text boxes) so the user can edit the Person objects. I have to switch from "text" to the Knockout "value" keyword to enable two-way binding to user input. I also add an image with a click event bound to the PeopleViewModel.save method:

```
<img src="../../Images/save.png" data-bind="click: save" title="Save Changes" />
<ul data-bind="foreach: people">
  <li class="person" >
    <form>
      <label>First: </label><input data-bind="value: FirstName" />
      <label>Last: </label> <input data-bind="value: LastName" />
    </form>
  </li>
</ul>
```

That's it. Breeze and Knockout will take care of the rest! You can see the data displayed for editing in **Figure 4**.

I can edit any or all of these fields and click the save button. The Breeze EntityManager will gather up all of the data changes and push them up to the server, which in turn will send them to Entity Framework to update the database. While I won't be extending this demo to include inserts and deletes, Breeze can certainly handle those modifications as well.

And for the Big Finish—Adding in Related Data

This is the part of any JavaScript application that many developers dread—and it's exactly the reason I wanted to write this column.

I'll make one small change to my script and add a little markup to the form that will turn each person into an editable master/detail person.

The change in my script will be in dataservice, where I'll modify the query by adding in the Breeze expand query method to eager-load each person's devices along with the person. Expand is a term you might recognize from OData or NHibernate, and is similar to Include in the Entity Framework (Breeze also has support to easily load related data after the fact):

```
var query = new entityModel.EntityQuery()
          .from("People")
          .expand("Devices")
          .orderBy("FirstName, LastName");
```

I'll then modify my view so that it knows how to display the Device data, as shown in **Figure 5**.

And there you have it. As you can see in **Figure 6**, Breeze handles the eager loading and building of the graphs on the client side. It also coordinates the data to be sent back to the service for updates. On the server side, the Breeze EFContextProvider sorts out all of the change data that it receives and makes sure that the Entity Framework gets what it needs to persist the data to the database.

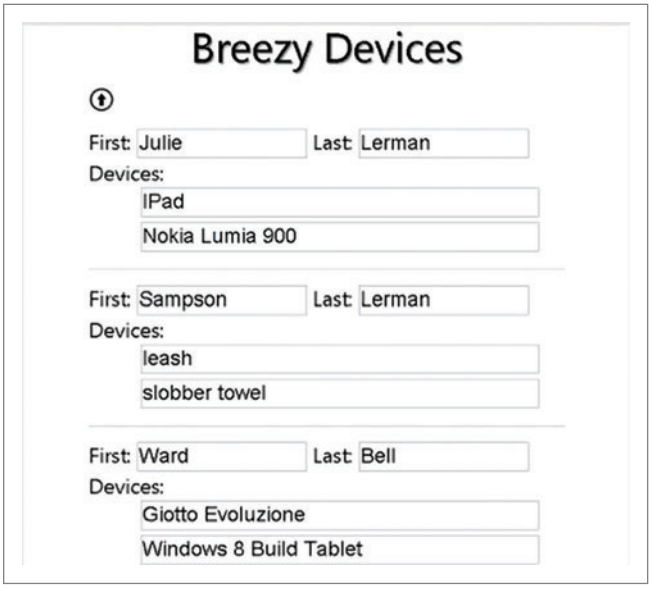


Figure 6 **Consuming and Saving Related Data**

While this was simple with a one-to-many relationship, at the time of this writing the beta version of Breeze doesn't support many-to-many relationships.

> While I won't be extending this demo to include inserts and deletes, Breeze can certainly handle those modifications.

Pain-Free Client-Side Data Access

Bell tells me it was his own painful experience of working on a project that was both JavaScript-intensive and data-access-intensive that inspired Breeze. His company, IdeaBlade, has always focused on creating solutions to solve the problems of handling disconnected data, and the developers were able to bring a great deal of experience to this open source project. I've always been reluctant to embark on projects that use a lot of JavaScript because my skills are lame to begin with, and I know the data-access bits would make me unhappy. I was very interested in Breeze as soon as I saw it. And though I've only scratched its surface, the last bit of what I showed you in this article—how easy it was to consume and save related data—is what really won me over. ■

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

Thanks to the following technical expert for reviewing this article: *Ward Bell*

BRUNO TERKALY AND
RICARDO VILLALOBOS

Windows Azure Service Bus: Messaging Patterns Using Sessions

In one of our previous articles, we discussed the importance of using messaging patterns in the cloud in order to decouple solutions and promote easy-to-scale software architectures. (See "Comparing Windows Azure Queues and Service Bus Queues" at msdn.microsoft.com/magazine/jj159884.) Queuing is one of these messaging patterns, and the Windows Azure platform offers two main options to implement this approach: Queue storage services and Service Bus Queues, both of which cover scenarios where multiple consumers compete to receive and process each of the messages in a queue. This is the canonical model for supporting variable workloads in the cloud, where receivers can be dynamically added or removed based on the size of the queue, offering a load balancing/failover mechanism for the back end (see **Figure 1**).

Even though the queuing messaging pattern is a great solution for simple decoupling, there are situations where each receiver requires its own copy of the message, with the option of discarding some messages based on specific rules. A good example of this type of scenario is shown in **Figure 2**, which illustrates a common challenge that retail companies face when sending information to multiple branches, such as the latest products catalog or an updated price list.

For these situations, the publisher/subscriber pattern is a better fit, where the receivers simply express an interest in one or more message categories, connecting to an independent subscription that contains a copy of the message stream. The Windows Azure Service Bus implements the publisher/subscriber messaging pattern through topics and subscriptions, which greatly enhances the ability to control how messages are distributed, based on independent rules and filters. In this article, we'll explain how to apply these Windows Azure Service Bus capabilities using a simple real-life scenario, assuming the following requirements:

1. Products should be received in order, based on the catalog page.
2. Some of the stores don't carry specific catalog categories, and products in these categories should be filtered out for each store.
3. New catalog information shouldn't be applied to the store system until all the messages have arrived.

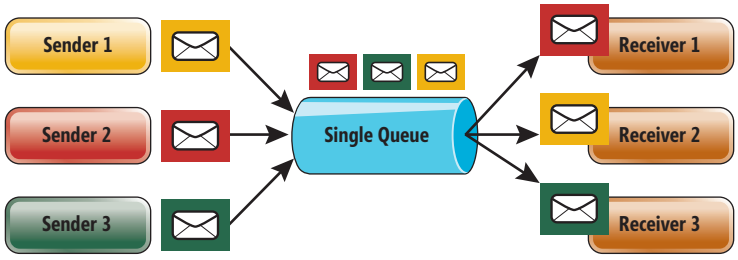


Figure 1 **Queuing Messaging Pattern: Each Message Is Consumed by a Single Receiver**

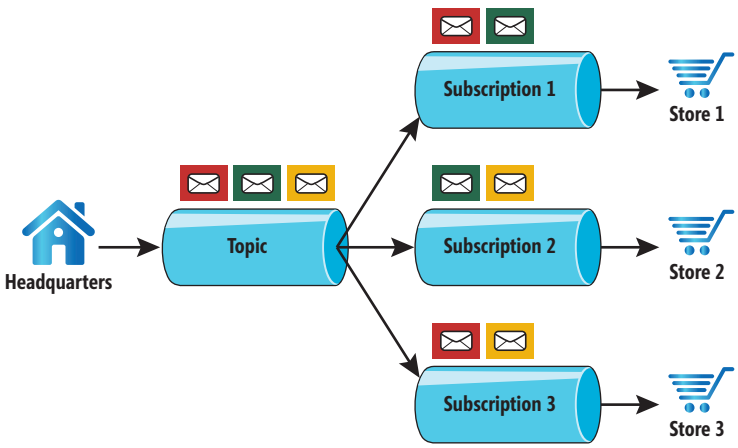


Figure 2 **Publisher/Subscriber Messaging Pattern: Each Message Can Be Consumed More Than Once**

Code download available at archive.msdn.microsoft.com/mag201212AzureInsider.

All the code samples for this article were created with Visual Studio 2012, using C# as the programming language. You'll also need the Windows Azure SDK version 1.8 for .NET developers and access to a Windows Azure subscription.

Setting up the Messaging Blueprint for the Project

Before writing any code, you need to define the different entities (topics and subscriptions) that will become part of the messaging workflow. This can be accomplished by accessing the Windows Azure Portal at manage.windowsazure.com. Log in with your credentials and follow these steps:

1. Click the Create New icon on the bottom left of the Management Portal.

Aspose.Total just got BIGGER

Aspose.Diagram

Working with Visio files? Easily create, modify and convert diagrams in your applications.

Supported Files

VSD VTX
VSS VDW
VST VDX
VSX

NEW

Aspose.OCR

Extract text from images. Supports popular fonts and styles. Scan a whole image or part of an image.

Supported Files

BMP
TIFF

NEW

Aspose.Imaging

Add advanced drawing features to your applications, plus support for PSD files.

Supported Files

PSD BMP
TIFF PNG
JPEG
GIF

NEW

Already own Aspose.Total for .NET?

These are yours for FREE!

Free Evaluations at www.aspose.com

EU Sales: +44 (0) 141 416 1112
sales.europe@aspose.com
AU Sales: +61 2 8003 5926
sales.asiapacific@aspose.com

US Sales: 1.888.277.6734
sales@aspose.com

ASPOSE
Your File Format Experts

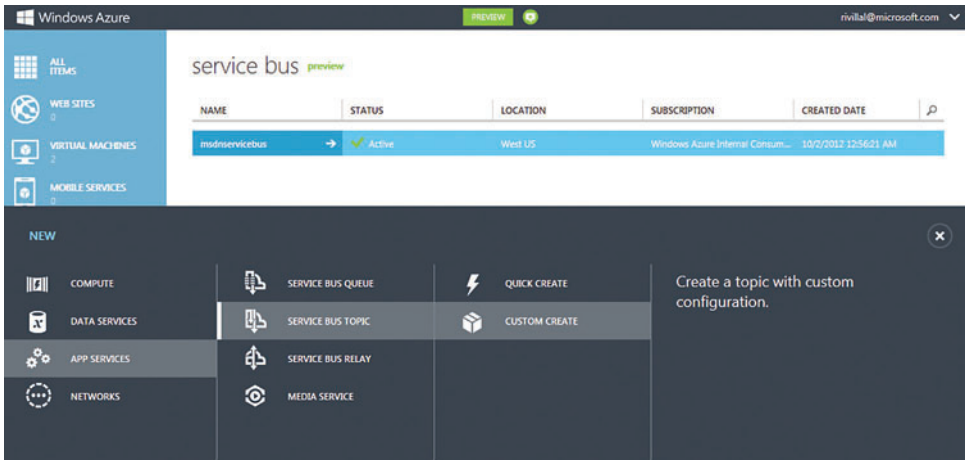


Figure 3 **Creating a New Service Bus Topic Using the Windows Azure Portal**

2. Click on the APP SERVICES icon, then on SERVICE BUS TOPIC and finally on CUSTOM CREATE (see **Figure 3**).
3. On the first dialog screen, enter the topic name and select the appropriate region and Windows Azure subscription ID. If this is your first namespace in the selected region, the wizard will suggest a namespace queue: [your entity name]-ns. You can change this value.
4. Click on the NEXT mark (right-pointing arrow) to insert the remaining properties. You can keep the default values. Click on the checkmark to create the topic.
5. Click on the Service Bus icon on the left navigation bar to get a list of namespaces. Note that you might not see the namespace listed immediately. It takes a few seconds to create the namespace and update the portal interface.
6. Select the Topic that you just created from the list, and click on ACCESS KEY, which can be found on the bottom of the screen. Record the full connection string for later use.
7. On the top of the Windows Azure Portal screen, click on SUBSCRIPTIONS, and then on CREATE A NEW SUBSCRIPTION. In the popup dialog, enter a name (in our example, we used "Store1Sub"), and click on the arrow to continue.
8. In the next screen, keep the default values, but make sure to check the Enable sessions option. Click on the checkmark to create the subscription. Sessions will be used by subscribers to retrieve messages in sequential order.
9. Repeat steps 7 and 8 for each of the three stores.

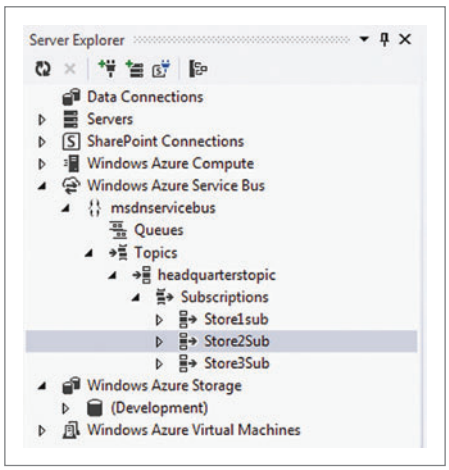


Figure 4 **Creating a Service Bus Topic and Subscriptions Using the Visual Studio Tools**

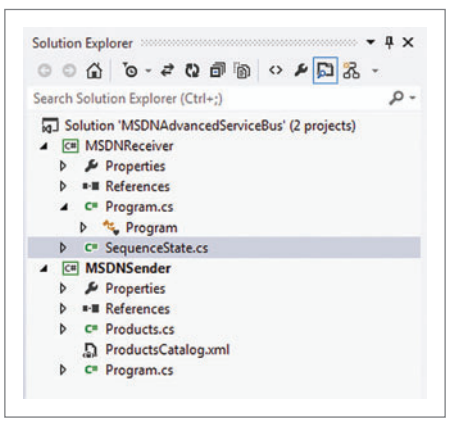


Figure 5 **Visual Studio Solution to Simulate the Products Catalog Scenario**

Once the topics and subscriptions have been created, you can also access them directly in Visual Studio. To do so, open Server Explorer (View | Server Explorer) and expand the Windows Azure Service Bus node (see **Figure 4**). Right-click on the Windows Azure Service Bus node and select Add New Connection. Enter the Namespace name, Issuer Name (usually "owner") and Issuer Access Key you recorded when the Windows Azure namespace was created in the portal.

Keep in mind that it's possible to programmatically create and manage these entities using classes in the Microsoft.ServiceBus.Messaging namespace, including TopicClient and SubscriptionClient, which are used later in this article.

Once the basic structure for the messaging workflow has been created, we'll simulate traffic using two console applications created in Visual Studio, as shown in **Figure 5**. The first console application, MSDNSender, will send the products catalog. The second, MSDNReceiver, will receive the information in each of the stores. We'll analyze the code in the following sections. In the Pub/Sub pattern, the MSDNSender is the publisher and the MSDNReceiver is the subscriber.

Sending the Products Catalog from Headquarters

As you can see in **Figure 2**, Headquarters (the publisher) sends messages to a topic. This logic is represented by the code in the main file, Program.cs, a part of the MSDNSender project. Program.cs encapsulates the logic and code to send a list of products as individual messages to the topic. Let's take a look at the different sections, starting with the Main method. Notice that first we create a client for the topic, as follows:

```
// Create a topicClient using the
// Service Bus credentials
TopicClient topicClient =
  TopicClient.CreateFromConnectionString(
  serviceBusConnectionString, topicName);
```

Once a topicClient is created, the publisher can send messages using it. The list of products to be sent is stored in an XML file called ProductsCatalog.xml, which contains a list of 10 product entities that will be transformed into an array of objects. The products will then get mapped into the Catalog and Product classes stored in the Product.cs file:

componentart.com/windows8

Figure 6 **Class Representation for Products in the Catalog**

```
public class Product
  {
    [System.Xml.Serialization.XmlElement("ProductId")]
    public string ProductId { get; set; }

    [System.Xml.Serialization.XmlElement("ProductName")]
    public string ProductName { get; set; }

    [System.Xml.Serialization.XmlElement("Category")]
    public string Category { get; set; }

    [System.Xml.Serialization.XmlElement("CatalogPage")]
    public int CatalogPage { get; set; }

    [System.Xml.Serialization.XmlElement("MSRP")]
    public double MSRP { get; set; }

    [System.Xml.Serialization.XmlElement("Store")]
    public string Store { get; set; }
  }
```

```
// Deserialize XML file with Products, and store them in an object array
Catalog catalog = null;
string path = "ProductsCatalog.xml";
XmlSerializer serializer = new XmlSerializer(typeof(Catalog));
StreamReader reader = new StreamReader(path);
catalog = (Catalog) serializer.Deserialize(reader);
reader.Close();
```

Each Product in the catalog array presents the structure shown in **Figure 6**.

> Sessions are extremely important, because they allow the receiver to determine whether all the messages that belong to a specific logical group have arrived.

Inside the array loop, a call to the CreateMessage method extracts different properties from the Product objects and assigns them to the message to be sent. Two properties require extra attention:

```
if (isLastProductInArray)
  message.Properties.Add("IsLastMessageInSession", "true");
message.SessionId = catalogName;
```

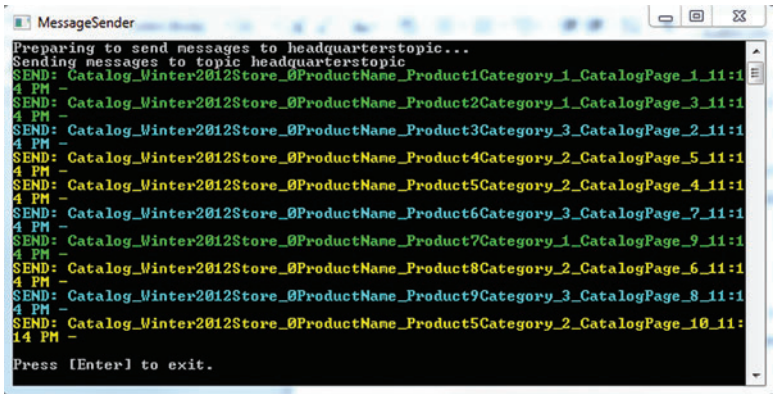


Figure 7 **Execution of the MSDNSender Project**

Figure 8 **The ReceivedSessionMessages Method in Code**

```
static void ReceiveSessionMessages(MessageSession receiver)
  {
    // Read messages from subscription until subscription is empty
    Console.WriteLine("Reading messages from subscription {0}", receiver.Path);
    Console.WriteLine("Receiver Type:" + receiver.GetType().Name);
    Console.WriteLine("Receiver.SessionId = " + receiver.SessionId);

    SequenceState sessionState = GetState(receiver);
    BrokeredMessage receivedMessage;
    while ((receivedMessage = receiver.Receive()) != null)
    {
      string sessionId = receiver.SessionId;
      ProcessMessage(receivedMessage, ref sessionState, receiver);
      while (sessionState.GetNextOutOfSequenceMessage() != -1)
      {
        // Call back deferred messages
        Console.WriteLine("Calling back for deferred message: Category {0},
          Message sequence {1}", receiver.SessionId,
            sessionState.GetNextSequenceId());
        receivedMessage = receiver.Receive(
          sessionState.GetNextOutOfSequenceMessage());
        ProcessMessage(receivedMessage, ref sessionState, receiver);
      }

      if (receivedMessage.Properties.ContainsKey(
        "IsLastMessageInSession"))
        break;

    }

    SetState(receiver, null);
    receiver.Close();
  }
```

Sessions are extremely important, because they allow the receiver to determine whether all the messages that belong to a specific logical group have arrived. In this case, by setting the SessionId message property, we're specifying that the receiver shouldn't use the catalog information until after all the messages with the same catalogName value have arrived. Also, for the last product in the array, we're adding a new property: IsLastMessageInSession, which will allow the receivers to determine if the last message in the session has arrived, and the catalog can be fully processed. **Figure 7** shows MSDNSender running.

Receiving the Product Catalog Using Subscriptions at the Stores

Now that the catalog and products have been sent out to the topic and copied to the different subscriptions, let's turn our attention to the MSDNReceiver project, where messages are received and processed. Note that in the Main method of Program.cs, the code creates a client for the Subscription based on information provided by the user via a Console.ReadLine command. Users are expected to enter their store number, which reflects the messages they wish to receive. In short, each branch store is concerned only with messages that apply to that store:

```
Console.WriteLine("Enter Store Number");
string storeNumber = Console.ReadLine();

Console.WriteLine("Selecting Subscription for Store...");

// Create a Subscription Client to the Topic
SubscriptionClient subscriptionClient =
  SubscriptionClient.CreateFromConnectionString(
  serviceBusConnectionString, topicName,
  "Store" + storeNumber.Trim() + "Sub",
  ReceiveMode.PeekLock);
```

componentart.com/windows8

Figure 9 **The ProcessMessage Method in Code**

```
static void ProcessMessage(BrokeredMessage message, ref SequenceState sessionState,
  MessageSession session = null)
  {
    if (session != null)
    {
      int messageId = Convert.ToInt32(message.Properties["CatalogPage"]);
      if (sessionState.GetNextSequenceId() == messageId)
      {
        OutputMessageInfo("RECV: ", message, "State: " + "RECEIVED");
        sessionState.SetNextSequenceId(messageId + 1);
        message.Complete();
        SetState(session, sessionState);
      }
      else
      {
        Console.WriteLine("Deferring message: Category {0}, Catalog Page {1}",
          session.SessionId, messageId);
        sessionState.AddOutOfSequenceMessage(messageId, message.SequenceNumber);
        message.Defer();
        SetState(session, sessionState);
      }
    }

    Thread.Sleep(receiverDelay);
  }
```

Because we're receiving messages from the subscriptions based on sessions (as explained in the previous section), we need to request the next one using the following line of code:

```
MessageSession sessionReceiver =
  subscriptionClient.AcceptMessageSession(TimeSpan.FromSeconds(5));
```

Basically, what this means is that the client will check for any to-be-processed messages in the subscription—those whose SessionId property is not null—and if no such messages are encountered within a period of five seconds, the request will time out, terminating the receiver application. On the other hand, if a session is found, the ReceivingSessionMessages method will be called. Before we jump into this piece of code, let's discuss the concept of session state, which allows the developer to store information that can be used while messages that belong to the same transaction are received. In this case, we're using session state to "remember" the last catalog page that was received, as well as the messages—products—that arrived out of order.

Based on this, here's the workflow in code:

1. The current message is received in the ReceiveSessionMessages method (see **Figure 8**), which relies on the ProcessMessage method (**Figure 9**) to process it.
2. Inside the ProcessMessage method, if the message is out of sequence, it's automatically deferred and its ID stored in the session state. Otherwise, it's marked as "complete" and removed from the subscription. Also, the next expected sequence—Catalog Page—is stored in the session.
3. After the current received message has been processed, the subsequent code in ReceiveSessionMessages checks for deferred message IDs in the session, and tries to process them again based on the latest Catalog page.
4. Once all the messages have been received for the session, the receiver is closed.

Keep in mind that for this project, deferred message IDs are stored in the session state, and could be potentially lost. In a production environment, we recommend using some type of persisted storage (Windows Azure Tables is one option) for this purpose. Note that if the message contains the property IsLastMessageSessionInSession (set during the sending process), the session loop is terminated. The console output for the MSDNReceiver project can be seen in **Figure 10**.

Windows Azure Service Bus subscriptions give you the ability to create specific rules that filter out messages before they're consumed. In this case, it would be relatively easy to create a rule that segregates products by category or by store number (which we ignored in this project). Rules can be created programmatically, directly in the Windows Azure portal or through Visual Studio tools.

Wrapping Up

The Windows Azure Service Bus offers an amazingly robust and flexible implementation of the publish/subscribe pattern. Many different scenarios can be addressed through the use of topics and subscriptions. The ability to support multiple senders broadcasting messages to multiple receivers, combined with the ability to logically group and sort messages, opens up a world of possibilities for the modern developer. Moreover, being able to leverage a persistent session to track state makes it straightforward to logically group messages and control their sequence. In a world where distributed environments are the norm, understanding how to use messaging patterns and the tools around them is crucial for today's software architects working in the cloud. ■

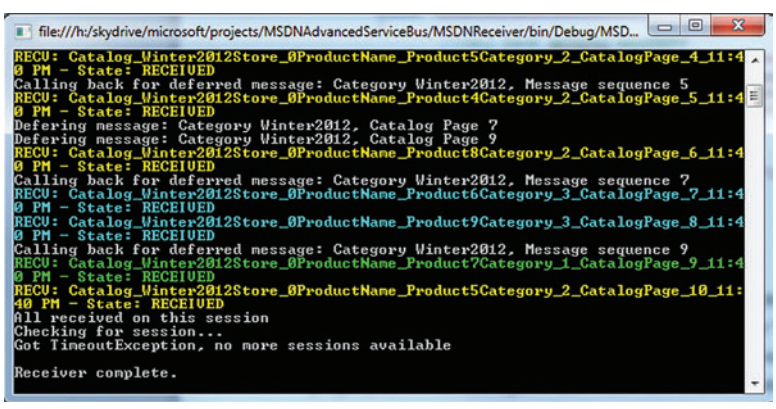


Figure 10 **Execution of the MSDNReceiver project**

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.*

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 cloud architect in the Windows Azure CSV incubation group for Microsoft.*

Thanks to the following technical expert for reviewing this article: *Abhishek Lal*

5 YEARS OF EXCELLENCE

XCEED
DataGrid
for WPF

Mature, feature-packed, and lightning-fast. The most adopted and trusted WPF datagrid.

Microsoft®
Visual Studio® Team System 2010

"Using Xceed DataGrid for WPF in Microsoft Visual Studio System 2010 helped us greatly reduce the time and resources necessary for developing all the data presentation features we needed. Working with Xceed has been a pleasure."

Norman Guadagno
Director of Product Marketing
for Microsoft Visual Studio Team System

XCEED
DataGrid
for Silverlight

Incredible streaming technology. Speed up your app and say goodbye to paging.

XCEED
Ultimate ListBox
for WPF

The world's first streaming listbox. Simple, drop-in upgrade to the WPF listbox.

XCEED
Ultimate ListBox
for Silverlight

Fast and fluid, with ground-breaking streaming technology.

Microsoft Visual Studio PARTNER

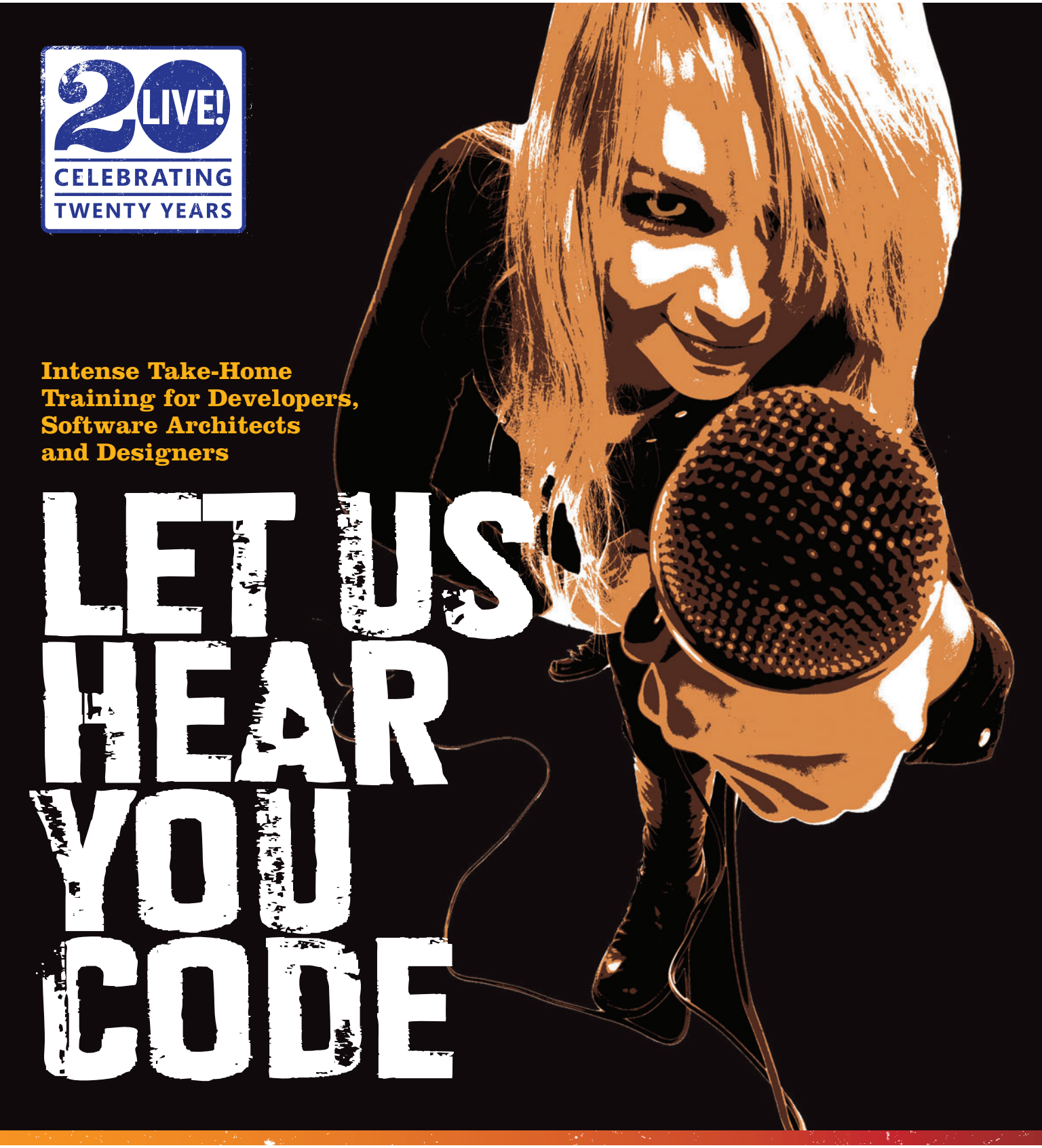
XCEED
MULTI-TALENTED COMPONENTS

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

YOUR BACKSTAGE PASS TO THE MICROSOFT PLATFORM

20 LIVE! CELEBRATING TWENTY YEARS



Intense Take-Home Training for Developers, Software Architects and Designers

LET US HEAR YOU CODE

SUPPORTED BY Microsoft Visual Studio msdn magazine Visual Studio MAGAZINE

PRODUCED BY 1105 MEDIA

LAS VEGAS | MARCH 25-29, 2013

MGM Grand Hotel & Casino

END-OF-YEAR SPECIAL:

Register before January 1 and save $500!

Use Promo Code LVDEC

Everyone knows all the *really* cool stuff happens behind the scenes. Get an all-access look at the Microsoft Platform and practical, unbiased, Developer training at Visual Studio Live! Las Vegas.

Code with .NET rockstars and learn how to maximize the development capabilities of Visual Studio and .NET during 5 action-packed days of pre- and post-conference workshops, 60+ sessions led by expert instructors and keynotes by industry heavyweights.

Topics will include:

- Cloud Computing
- Data Management
- HTML5
- Mobile
- SharePoint
- Silverlight / WPF
- SQL Server
- Visual Studio / .NET
- Web
- Windows 8
- Windows Phone

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

vslive.com/lasvegas

Windows 8 and the WebSocket Protocol

Kenny Kerr

The WebSocket protocol aims to provide bidirectional communication in a Web-saturated world dominated by clients solely responsible for establishing connections and initiating request/response pairs. It finally allows applications to enjoy many more of the benefits of TCP, but in a Web-friendly way. Considering that the WebSocket protocol was only standardized by the Internet Engineering Task Force in December 2011—and as I write this is still under consideration by the World Wide Web Consortium—it's perhaps surprising just how comprehensively Windows 8 has embraced this new Internet technology.

This article discusses:

- The TCP/IP suite
- TCP and HTTP
- The WebSocket handshake
- WebSocket data transfer
- Windows 8 and the WebSocket protocol
- .NET clients and servers
- Support by the Windows Runtime
- JavaScript and the WebSocket protocol
- The Windows HTTP Services Client for C++

Technologies discussed:

Windows 8, WebSocket Protocol

In this article I'll first show you how the WebSocket protocol works and explain its relationship to the larger TCP/IP suite. I'll then explore the various ways in which Windows 8 enables programmers to easily adopt this new technology from within their applications.

Why WebSocket?

The main goal of this protocol is to provide a standard and efficient way for browser-based applications to communicate with servers freely outside of request/response pairs. A few years back, Web developers were all aflutter talking about Asynchronous JavaScript and XML (AJAX) and how it enables dynamic and interactive scenarios—and certainly it did, but the XMLHttpRequest object that inspired it all still only allowed the browser to make HTTP requests. What if the server wanted to send a message to the client out-of-band? That's where the WebSocket protocol comes in. It not only allows the server to send messages to the client, but it does so without the overhead of HTTP, providing bidirectional communication that's close to the speed of a raw TCP connection. Without the WebSocket protocol, Web developers have had to abuse HTTP by polling the server for updates, using Comet-style programming techniques, and employing many HTTP connections with a great deal of protocol overhead just to keep applications up-to-date. Servers are overloaded, bandwidth is wasted and Web applications are overly complicated. The WebSocket protocol solves these problems in a surprisingly simple and efficient way, but before I can describe how it works, I need to provide some foundational and historical context.

The TCP/IP Suite

TCP/IP is a protocol suite, or collection of interrelated protocols, that implements the Internet architecture. It has evolved into its current form over many years. The world has changed dramatically since the 1960s, when the concept of packet-switching networks first developed. Computers have become much faster, software has grown more demanding and the Internet has exploded into an all-encompassing web of information, communication and interaction, and is the backbone of so much of the software in popular use today.

The TCP/IP suite consists of a number of layers loosely modeled after the Open System Interconnection (OSI) layering model. Although the protocols at the different layers aren't particularly well delineated, TCP/IP has clearly proven its effectiveness, and the layering problems have been overcome by a clever combination of hardware and software designs. Separating TCP/IP into layers, however vague they might be, has helped it evolve over time as hardware and technology have changed, and has allowed programmers with different skills to work at different levels of abstraction, either helping to build the protocol stack itself or to build applications making use of its various facilities.

At the lowest layers are the physical protocols, including the likes of wired media access control and Wi-Fi, providing physical connectivity as well as local addressing and error detection. Most programmers don't think too much about these protocols.

Moving up the stack, the Internet Protocol (IP) itself resides in the networking layer and allows TCP/IP to become interoperable across different physical layers. It takes care of mapping computer addresses to physical addresses and routing packets from computer to computer.

Then there are ancillary protocols, and we could debate about which layer they reside on, but they really provide a necessary supporting role for things such as auto-configuration, name resolution, discovery, routing optimizations and diagnostics.

As we move further up the layering stack, the transport and application protocols come into view. The transport protocols take care of multiplexing and de-multiplexing the packets from the lower layers so that, even though there might only be a single physical and networking layer, many different applications can share the communication channel. The transport layer also typically provides further error detection, reliable delivery and even performance-related features such as congestion and flow control. The application layer has traditionally been the home of protocols such as HTTP (implemented by Web browsers and servers) and SMTP (implemented by e-mail clients and servers). As the world has started relying more heavily on protocols such as HTTP, their implementations have been pushed down into the depths of the OS, both to improve performance as well as to share the implementation among different applications.

TCP and HTTP

Of the protocols in the TCP/IP suite, the TCP and User Datagram Protocol (UDP) found at the transport layer are perhaps the most well known to the average programmer. Both define a "port" abstraction that these protocols use in combination with IP addresses to multiplex and de-multiplex packets as they arrive and when they're sent.

Although UDP is used heavily for other TCP/IP protocols such as Dynamic Host Configuration Protocol and DNS, and has been adopted widely for private network applications, its adoption in the Internet at large hasn't been as far-reaching as that of its sibling. TCP, on the other hand, has seen widespread adoption across the board, thanks in large part to HTTP. Although TCP is far more complex than UDP, much of this complexity is hidden from the application layer where the application enjoys the benefits of TCP without being subject to its complexity.

TCP provides a reliable flow of data between computers, the implementation of which is hugely complex. It concerns itself with packet ordering and data reconstruction, error detection and recovery, congestion control and performance, timeouts, retransmissions, and much more. The application, however, only sees a bidirectional connection between ports and assumes that data sent and received will transfer correctly and in order.

Contemporary HTTP presupposes a reliable connection-oriented protocol, and TCP is clearly the obvious and ubiquitous choice. In this model, HTTP functions as a client-server protocol. The client opens a TCP connection to a server. It then sends a request, to which the server evaluates and responds. This is repeated countless times every second of every day around the world.

Of course, this is a simplification or restriction of the functionality that TCP provides. TCP allows both parties to send data simultaneously. One end doesn't need to wait for the other to send a request before it can respond. This simplification did, however, allow server-side caching of responses, which has had a huge impact on the Web's ability to scale. But the popularity of HTTP was undoubtedly aided by its initial simplicity. Whereas TCP provides a bidirectional channel for binary data—a pair of streams, if you like—HTTP provides a request message preceding a response message, both consisting of ASCII characters, although the message bodies, if any, may be encoded in some other way. A simple request might look as follows:

```
GET /resource HTTP/1.1\r\n
host: example.com\r\n
\r\n
```

Each line concludes with a carriage return (\r) and line feed (\n). The first line, called a request line, specifies the method by which a resource is to be accessed (in this case GET), the path of the resource and finally the version of HTTP to be used. Similar to the lower-layer protocols, HTTP provides multiplexing and de-multiplexing via this resource path. Following this request line are one or more header lines. Headers consist of a name and value as illustrated in the preceding example. Some headers are required, such as host, while most are not and merely assist browsers and servers in communicating more efficiently or to negotiate features and functionality.

A response might look like this:

```
HTTP/1.1 200 OK\r\n
content-type: text/html\r\n
content-length: 1307\r\n
\r\n
<!DOCTYPE HTML><html> ... </html>
```

The format is basically the same, but instead of a request line, the response line affirms the version of HTTP to be used, a status code (200) and a description of the status code. The 200 status code indicates to the client that the request was processed successfully

and any result is included immediately following any header lines. The server might, for example, indicate that the requested resource doesn't exist by returning a 404 status code. The headers take the same form as those in the request. In this case the content-type header informs the browser that the requested resource in the message body is to be interpreted as HTML and the content-length header tells the browser how many bytes the message body contains. This is important because, as you'll recall, HTTP messages flow over TCP, which doesn't provide message boundaries. Without a content length, HTTP applications need to use various heuristics to determine the length of any message body.

This is all pretty simple, a testament to the straightforward design of HTTP. But HTTP isn't simple anymore. Today's Web browsers and servers are state-of-the-art programs with thousands of interrelated features, and HTTP is the workhorse that needs to keep up with it all. Much of the complexity was born out of a need for speed. There are now headers to negotiate compression of the message body, caching and expiration headers to avoid transmitting a message body at all, and much more. Techniques have been developed to reduce the number of HTTP requests by combining different resources. Content delivery networks (CDNs) have even been distributed around the world in an attempt to host commonly accessed resources closer to the Web browsers accessing them.

Despite all of these advances, many Web applications could achieve greater scalability and even simplicity if there were some way to occasionally break out of HTTP and return to the streaming model of TCP. This is exactly what the WebSocket protocol delivers.

The WebSocket Handshake

The WebSocket protocol fits somewhat neatly into the TCP/IP suite above TCP and alongside HTTP. One of the challenges with introducing a new protocol to the Internet is in somehow making the countless routers, proxies and firewalls think that nothing has changed under the sun. The WebSocket protocol achieves this goal by masquerading as HTTP before switching to its own WebSocket data transfer on the same underlying TCP connection. In this way, many unsuspecting intermediaries don't have to be upgraded in order to allow WebSocket communication to traverse their network connections. In practice this doesn't always work quite so smoothly because some overly zealous routers fiddle with the HTTP requests and responses, attempting to rewrite them to suit their own ends, such as proxy caching or address or resource translation. An effective solution in the short term is to use the WebSocket protocol over a secure channel—Transport Layer Security (TLS)—because this tends to keep the tampering to a minimum.

The WebSocket protocol borrows ideas from a variety of sources, including IP, UDP, TCP and HTTP, and makes those concepts available to Web browsers and other applications in a simpler form. It all starts with a handshake that's designed to look and operate just like an HTTP request/response pair. This isn't done so that clients or servers can somehow fool each other into using WebSockets, but rather to fool the various intermediaries into thinking it's just another TCP connection serving up HTTP. In fact, the WebSocket protocol is specifically designed to prevent any party from being duped into accepting a connection accidentally. It begins with a client sending a handshake that is, for all intents and purposes, an HTTP request, and might look as follows:

```
GET /resource HTTP/1.1\r\n
host: example.com\r\n
upgrade: websocket\r\n
connection: upgrade\r\n
sec-websocket-version: 13\r\n
sec-websocket-key: E4WSEcseoWr4csPLS2QJHA==\r\n
\r\n
```

As you can see, nothing precludes this from being a perfectly valid HTTP request. An unsuspecting intermediary should simply pass this request along to the server, which may even be an HTTP server doubling as a WebSocket server. The request line in this example specifies a standard GET request. This also means that a WebSocket server might allow multiple endpoints to be serviced by a single server in the same way that most HTTP servers do. The host header is required by HTTP 1.1 and serves the same purpose—to ensure both parties agree on the hosting domain in shared hosting scenarios. The upgrade and connection headers are also standard HTTP headers used by clients to request an upgrade of the protocol used in the connection. This technique is sometimes used by HTTP clients to transition to a secure TLS connection, although that's rare. These headers are, however, required by the WebSocket protocol. Specifically, the upgrade header indicates that the connection should be upgraded to the WebSocket protocol and the connection header specifies that this upgrade header is connection-specific, meaning that it must not be communicated by proxies over further connections.

The sec-websocket-version header must be included and its value must be 13. If the server is a WebSocket server but doesn't support this version, it will abort the handshake, returning an appropriate HTTP status code. As you'll see in a moment, even if the server knows nothing of the WebSocket protocol and happily returns a success response, the client is designed to abort the connection.

The sec-websocket-key header really is the key to the WebSocket handshake. The designers of the WebSocket protocol wanted to ensure that a server couldn't possibly accept a connection from a client that was not in fact a WebSocket client. They didn't want a malicious script to construct a form submission or use the XMLHttpRequest object to fake a WebSocket connection by adding the sec-* headers. To prove to both parties that a legitimate connection is being established, the sec-websocket-key header must also be present in the client handshake. The value must be a randomly selected—ideally cryptographically random—16-byte number, known as a nonce in security parlance, which is then base64-encoded for this header value.

Once the client handshake is sent, the client waits for a response to validate that the server is indeed willing and able to establish a WebSocket connection. Assuming the server doesn't object, it might send a server handshake as an HTTP response as follows:

```
HTTP/1.1 101 OK
upgrade: websocket\r\n
connection: upgrade\r\n
sec-websocket-accept: 7eQChgCtQMnVILefJAO6dK5JwPc=\r\n
\r\n
```

Again, this is a perfectly valid HTTP response. The response line includes the HTTP version followed by the status code, but instead of the regular 200 code indicating success, the server must respond with the standard 101 code indicating that the server

understands the upgrade request and is willing to switch protocols. The English description of the status code makes absolutely no difference. It might be "OK" or "Switching to WebSocket" or even a random Mark Twain quote. The important thing is the status code and the client must ensure that it's 101. The server could, for example, reject the request and ask the client to authenticate using a 401 status code before accepting a WebSocket client handshake. A successful response must, however, include the upgrade and connection headers to acknowledge that the 101 status code specifically refers to a switch to the WebSocket protocol, again to avoid anyone being duped.

Finally, to validate the handshake, the client ensures that the sec-websocket-accept header is present in the response and its value is correct. The server needn't decode the base64-encoded value sent by the client. It merely takes this string, concatenates the string representation of a well-known GUID and hashes the combination with the SHA-1 algorithm to produce a 20-byte value that's then base64-encoded and used as the value for the sec-websocket-accept header. The client can then easily validate that the server did indeed do as required and there's then no doubt that both parties are consenting to a WebSocket connection.

If all goes well, at this point a valid WebSocket connection is established and both parties can communicate freely and simultaneously in both directions using WebSocket data frames. It's clear from studying the WebSocket protocol that it was designed after the Web insecurity apocalypse. Unlike most of its predecessors, the WebSocket protocol was designed with security in mind. The protocol also requires that the client include the origin header if the client is in fact a Web browser. This allows browsers to provide protection against cross-origin attacks. Of course, this only makes sense in the context of a trusted hosting environment such as that of a browser.

WebSocket Data Transfer

The WebSocket protocol is all about getting the Web back to the relatively high-performance, low-overhead model of communication provided by IP and TCP, not adding further layers of complexity and overhead. For this reason, once the handshake completes, the WebSocket overhead is kept to a minimum. It provides a packet-framing mechanism on top of TCP reminiscent of the IP packetization that TCP itself is built on and for which UDP is so popular, but without the packet size limitations with which those protocols are encumbered. Whereas TCP provides a stream-based abstraction, WebSocket provides a message-based abstraction to the application. And while TCP streams are transmitted via segments, WebSocket messages are transported as a sequence of frames. These frames are transmitted over the same TCP connection and thus naturally assume reliable and sequential delivery. The framing protocol is somewhat elaborate but is specifically designed to be extremely small, requiring in many cases only a few additional bytes of framing overhead. Data frames may be transmitted by either client or server at any time after the opening handshake has completed.

Each frame includes an opcode describing the frame type as well the size of the payload. This payload represents the actual data the application may want to communicate as well as any prearranged extension data. Interestingly, the protocol allows for messages to be fragmented. If you come from a hardcore networking background, you might be reminded of the performance implications of IP-level fragmentation and the pains to which TCP goes to avoid fragmentation. But the WebSocket concept of fragmentation is quite different. The idea here is to allow the WebSocket protocol to provide the convenience of network packets but without the size limits. If the sender doesn't know the exact length of a message being sent, it may be fragmented, with each frame indicating how much data it provides and whether or not it's the last fragment. Beyond that, the frame merely indicates whether it contains binary data or UTF-8-encoded text.

Control frames are also defined and primarily used to close a connection but can also be used as a heartbeat to ping the other endpoint to ensure it's still responsive or to assist in keeping the TCP connection alive. Finally, I should point out that if you happen to poke at a WebSocket frame sent by a client using a network protocol analyzer such as Wireshark, you might notice that the data frames appear to contain encoded data. The WebSocket protocol requires that all data frames sent from the client to the server be masked. Masking involves a simple algorithm "XOR'ing" the data bytes with a masking key. The masking key is contained within the frame, so this isn't meant to be some sort of ridiculous security feature, although it does relate to security. As mentioned, the designers of the WebSocket protocol spent a great deal of effort working through various security-related scenarios to try to anticipate the various ways in which the protocol might be attacked. One such attack vector that was analyzed involved attacking the WebSocket protocol indirectly by compromising other parts of the Internet's infrastructure, in this case proxy servers. Unsuspecting proxy servers that may not be aware of the WebSocket handshake's likeness to a GET request could be fooled into caching data for a fake GET request initiated by an attacker, in effect poisoning the cache for some users. Masking each frame with a new key mitigates this particular threat by ensuring that frames aren't predictable and thus can't be misconstrued on the wire. There's quite a bit more to this attack, and undoubtedly researchers will uncover further possible exploits in time. Still, it's impressive to see the lengths to which the designers have gone to try to anticipate many forms of attack.

Windows 8 and the WebSocket Protocol

As helpful as it is to have a deep understanding of the WebSocket protocol, it also helps a great deal to work on a platform with such wide-ranging support, and Windows 8 certainly delivers. Let's take a look at some of the ways in which you can use the WebSocket protocol without actually having to implement the protocol yourself.

Windows 8 provides the Microsoft .NET Framework, supports clients through the Windows Runtime for both native and managed code and lets you create WebSocket clients using the Windows HTTP Services (WinHTTP) API in C++. Finally, IIS 8 provides a native WebSocket module, and of course Internet Explorer provides native support for the WebSocket protocol. That's quite a mix of different environments, but what might be even more surprising is that Windows 8 only includes a single WebSocket implementation, which is shared among all of these. The WebSocket

Figure 1 **WebSocket Server Using HttpListener**

```
static async Task Run()
{
  HttpListener s = new HttpListener();
  s.Prefixes.Add("http://localhost:8000/ws/");
  s.Start();

  var hc = await s.GetContextAsync();

  if (!hc.Request.IsWebSocketRequest)
  {
    hc.Response.StatusCode = 400;
    hc.Response.Close();
    return;
  }

  var wsc = await hc.AcceptWebSocketAsync(null);
  var ws = wsc.WebSocket;

  for (int i = 0; i != 10; ++i)
  {
    await Task.Delay(2000);

    var time = DateTime.Now.ToLongTimeString();
    var buffer = Encoding.UTF8.GetBytes(time);
    var segment = new ArraySegment<byte>(buffer);

    await ws.SendAsync(segment, WebSocketMessageType.Text,
      true, CancellationToken.None);
  }

  await ws.CloseAsync(WebSocketCloseStatus.NormalClosure,
    "Done", CancellationToken.None);
}
```

Protocol Component API implements all of the protocol rules for handshaking and framing without ever actually creating a network connection of any kind. The different platforms and runtimes can then use this common implementation and hook it into the networking stack of their choice.

.NET Clients and Servers

The .NET Framework provides extensions to ASP.NET and provides HttpListener—which is itself based on the native HTTP Server API used by IIS—to provide server support for the WebSocket protocol. In the case of ASP.NET, you can simply write an HTTP handler that calls the new HttpContext.AcceptWebSocketRequest method to accept a WebSocket request on a particular endpoint. You can validate that the request is indeed a WebSocket client handshake using the HttpContext.IsWebSocketRequest property. Outside of ASP.NET, you can host a WebSocket server by simply using the HttpListener class. The implementation is also mostly shared between the two. **Figure 1** provides a simple example of such a server.

Here I'm using a C# async method to keep the code sequential and coherent, but in fact it's all asynchronous. I start by registering the endpoint and waiting for an incoming request. I then check whether the request does in fact qualify as a WebSocket handshake and return a 400 "bad request" status code if it isn't. I then call AcceptWebSocketAsync to accept the client handshake and wait for the handshake to complete. At this point, I can freely communicate using the WebSocket object. In this example the server sends 10 UTF-8 frames, each containing the time, after a short delay. Each frame is sent asynchronously using the SendAsync method. This method is quite powerful and can send UTF-8 or binary frames either as a whole or in fragments. The third parameter—in this case, true—indicates whether this call to SendAsync represents the end of the message. Thus, you can use this method repeatedly to send long messages that will be fragmented for you. Finally, the CloseAsync method is used to perform a clean closure of the WebSocket connection, sending a close control frame and waiting for the client to acknowledge with its own close frame.

On the client side, the new ClientWebSocket class uses an HttpWebRequest object internally to provide the ability to connect to a WebSocket server. **Figure 2** provides a simple example of a client that can be used to connect to the server in **Figure 1**.

Here I'm using the ConnectAsync method to establish a connection and perform the WebSocket handshake. Notice that the URL uses the new "ws" URI scheme to identify this as a WebSocket endpoint. As with HTTP, the default port for ws is port 80. The "wss" scheme is also defined to represent a secure TLS connection and uses the corresponding port 443. The client then calls ReceiveAsync in a loop to receive as many frames as the server is willing to send. Once received, the frame is first checked to see whether it represents a close control frame. In this case the client responds by sending its own close frame, allowing the server to close the connection promptly.

Figure 2 **WebSocket Client Using ClientWebSocket**

```
static async Task Client()
{
  ClientWebSocket ws = new ClientWebSocket();

  var uri = new Uri("ws://localhost:8000/ws/");
  await ws.ConnectAsync(uri, CancellationToken.None);
  var buffer = new byte[1024];

  while (true)
  {
    var segment = new ArraySegment<byte>(buffer);
    var result =
      await ws.ReceiveAsync(segment, CancellationToken.None);

    if (result.MessageType == WebSocketMessageType.Close)
    {
      await ws.CloseAsync(WebSocketCloseStatus.NormalClosure, "OK",
        CancellationToken.None);
      return;
    }

    if (result.MessageType == WebSocketMessageType.Binary)
    {
      await ws.CloseAsync(WebSocketCloseStatus.InvalidMessageType,
        "I don't do binary", CancellationToken.None);
      return;
    }

    int count = result.Count;

    while (!result.EndOfMessage)
    {
      if (count >= buffer.Length)
      {
        await ws.CloseAsync(WebSocketCloseStatus.InvalidPayloadData,
          "That's too long", CancellationToken.None);
        return;
      }

      segment =
        new ArraySegment<byte>(buffer, count, buffer.Length - count);
      result = await ws.ReceiveAsync(segment, CancellationToken.None);
      count += result.Count;
    }

    var message = Encoding.UTF8.GetString(buffer, 0, count);
    Console.WriteLine("> " + message);
  }
}
```

Compatible with
Microsoft® Visual Studio® 2012

FREE TRIAL DOWNLOAD
INFRAGISTICS.COM/DOWNLOADS

GET A PULSE

On Mobile Business Intelligence

infragistics.com/ **EXPERIENCE**

INFRAGISTICS
DESIGN / DEVELOP / EXPERIENCE

Infragistics Sales US 800 231 8588 • Europe +44 (0) 800 298 9055 • India +91 80 4151 8042 • APAC (+61) 3 9982 4545

Figure 3 **WebSocket Client Using the Windows Runtime**

```
static async Task Client()
{
  MessageWebSocket ws = new MessageWebSocket();
  ws.Control.MessageType = SocketMessageType.Utf8;

  ws.MessageReceived += (sender, args) =>
  {
    var reader = args.GetDataReader();
    var message = reader.ReadString(reader.UnconsumedBufferLength);
    Debug.WriteLine(message);
  };

  ws.Closed += (sender, args) =>
  {
    ws.Dispose();
  };

  var uri = new Uri("ws://localhost:8000/ws/");
  await ws.ConnectAsync(uri);
}
```

The client then checks whether the frame contains binary data, in which case it closes the connection with an error indicating that this frame type is unsupported. Finally, the frame data can be read. To accommodate fragmented messages, a while loop waits until the final fragment is received. The new ArraySegment structure is used to manage the buffer offset so the fragments are reassembled properly.

The WinRT Client

The Windows Runtime support for the WebSocket protocol is a little more restrictive. Only clients are supported, and fragmented UTF-8 messages must be completely buffered before they can be read. Only binary messages can be streamed with this API. **Figure 3** provides a simple example of a client that can also be used to connect to the server in **Figure 1**.

This example, although also written in C#, relies on event handlers for the most part, and the C# async method is of little utility, merely able to allow the MessageWebSocket object to connect asynchronously. The code is fairly simple, however, if a little quirky. The MessageReceived event handler is called once the entire (possibly fragmented) message is received and ready to read. Even though the entire message has been received and it can only ever be a UTF-8 string, it's stored in a stream, and a DataReader object must be used to read the contents and return a string. Finally, the Closed event handler lets you know that the server has sent a close control frame, but as with the .NET ClientWebSocket class, you're still responsible for sending a close control frame back to the server. The MessageWebSocket class, however, only sends this frame just before the object is itself destroyed. To make this happen promptly in C#, I need to call the Dispose method.

The Prototypical JavaScript Client

There's little doubt that JavaScript is the environment in which the WebSocket protocol will make the most impact, and the API is impressively simple. Here's all it takes to connect to the server in **Figure 1**:

```
var ws = new WebSocket("ws://localhost:8000/ws/");

ws.onmessage = function (args)
{
  var time = args.data;
  ...
};
```

Unlike the other APIs on Windows, the browser takes care of closing the WebSocket connection automatically when it receives a close control frame. You can, of course, explicitly close a connection or handle the onclose event, but no further action is required on your part to complete the closing handshake.

The WinHTTP Client for C++

Of course, the WinRT WebSocket client API can be used from native C++ as well, but if you're looking for a bit more control, then WinHTTP is just the thing for you. **Figure 4** provides a simple

Figure 4 **WebSocket Client Using WinHTTP**

```
auto s = WinHttpOpen( ... );

auto c = WinHttpConnect(s, L"localhost", 8000, 0);

auto r = WinHttpOpenRequest(c, nullptr, L"/ws/", ... );

WinHttpSetOption(r, WINHTTP_OPTION_UPGRADE_TO_WEB_SOCKET, nullptr, 0);

WinHttpSendRequest(r, ... );

VERIFY(WinHttpReceiveResponse(r, nullptr));

DWORD status;
DWORD size = sizeof(DWORD);

WinHttpQueryHeaders(r,
  WINHTTP_QUERY_STATUS_CODE | WINHTTP_QUERY_FLAG_NUMBER,
  WINHTTP_HEADER_NAME_BY_INDEX,
  &status,
  &size,
  WINHTTP_NO_HEADER_INDEX);

ASSERT(HTTP_STATUS_SWITCH_PROTOCOLS == status);

auto ws = WinHttpWebSocketCompleteUpgrade(r, 0);

char buffer[1024];
DWORD count;
WINHTTP_WEB_SOCKET_BUFFER_TYPE type;

while (NO_ERROR ==
  WinHttpWebSocketReceive(ws, buffer, sizeof(buffer), &count, &type))
{
  if (WINHTTP_WEB_SOCKET_CLOSE_BUFFER_TYPE == type)
  {
    WinHttpWebSocketClose(
      ws, WINHTTP_WEB_SOCKET_SUCCESS_CLOSE_STATUS, nullptr, 0);
    break;
  }

  if (WINHTTP_WEB_SOCKET_BINARY_MESSAGE_BUFFER_TYPE == type ||
    WINHTTP_WEB_SOCKET_BINARY_FRAGMENT_BUFFER_TYPE == type)
  {
    WinHttpWebSocketClose(
      ws, WINHTTP_WEB_SOCKET_INVALID_DATA_TYPE_CLOSE_STATUS, nullptr, 0);
    break;
  }

  std::string message(buffer, count);

  while (WINHTTP_WEB_SOCKET_UTF8_FRAGMENT_BUFFER_TYPE == type)
  {
    WinHttpWebSocketReceive(ws, buffer, sizeof(buffer), &count, &type);

    message.append(buffer, count);
  }

  printf("> %s\n", message.c_str());
}
```

example of using WinHTTP to connect to the server in **Figure 1**. This example is using the WinHTTP API in synchronous mode for conciseness, but this would work equally well asynchronously.

As with all WinHTTP clients, you need to create a WinHTTP session, connection and request object. There's nothing new here so I've elided some of the details. Before actually sending the request, you need to set the new WINHTTP_OPTION_UPGRADE_TO_WEB_SOCKET option on the request to instruct WinHTTP to perform a WebSocket handshake. The request is then ready to be sent with the WinHttpSendRequest function. The regular WinHttpReceiveResponse function is then used to wait for the response, which in this case will include the result of the WebSocket handshake. As always, to determine the result of a request, the WinHttpQueryHeaders function is called specifically to read the status code returned from the server. At this point, the WebSocket connection has been established and you can begin to use it directly. The WinHTTP API naturally handles the framing for you, and this functionality is exposed through a new WinHTTP WebSocket object that's retrieved by calling the WinHttpWebSocketCompleteUpgrade function on the request object.

Receiving the messages from the server is done, at least conceptually, in much the same way as the example in **Figure 2**. The WinHttpWebSocketReceive function waits to receive the next data frame. It also lets you read fragments of any kind of WebSocket message, and the example in **Figure 4** illustrates how this might be done in a loop. If a close control frame is received, then a matching close frame is sent to the server using the WinHttpWebSocketClose function. If a binary data frame is received, then the connection is similarly closed. Keep in mind that this only closes the WebSocket connection. You still need to call WinHttpCloseHandle to release the WinHTTP WebSocket object, as you have to do for all WinHTTP objects in your possession. A handle wrapper class such as the one I described in my July 2011 column, "C++ and the Windows API" (msdn.microsoft.com/magazine/hh288076), will do the trick.

The WebSocket protocol is a major new innovation in the world of Web applications and, despite its relative simplicity, is a welcome addition to the larger TCP/IP suite of protocols. I've little doubt that the WebSocket protocol will soon be almost as ubiquitous as HTTP itself, helping applications and connected systems of all kinds to communicate more easily and efficiently. Windows 8 has done its part to provide a comprehensive set of APIs for building both WebSocket clients and servers. ■

Kenny Kerr *is a software craftsman with a passion for native Windows development. Reach him at kennykerr.ca.*

Thanks to the following technical experts for reviewing this article: *Piotr Kulaga and Henri-Charles Machalani*

Speech-Enabling a Windows Phone 8 App, Part 2: In-App Dialog

F Avery Bishop

Last month, in part 1 (msdn.microsoft.com/magazine/jj721592) of this two-part series, I discussed enabling voice commands in a Windows Phone 8 app. Here, I'll discuss dialog with the user in a running app using speech input and output.

Once an app has been launched, many scenarios can benefit from interaction between the user and the phone using speech input and output. A natural one is in-app dialog. For example, the user can launch the Magic Memo app (see previous article) to go to the main page and then use speech recognition to enter a new memo, receive audio feedback and confirm the changes. Assuming no misrecognitions, the user can completely enter and save several memos without touching the phone (other than the first long push on the Start button).

This article discusses a prerelease version of Windows Phone 8. All related information is subject to change.

This article discusses:

- The speech synthesis API
- The speech recognition API
- Speech recognition grammars

Technologies discussed:

Windows Phone 8

Code download available at:

archive.msdn.microsoft.com/mag201211WP8Speech

You can imagine many other scenarios using speech dialog starting out in the app. For example, once the user has navigated to a page showing a list of saved favorites such as memos, movies or memorabilia, she could use recognition to choose one and take an action: edit, play, order, remove and so on. Speech output would then read back the selection and ask for confirmation.

In the following sections I'll lay out examples using speech for input and output, starting with simple examples and working up to more complex examples. I'll show how easy it is to implement

Figure 1 **Initializing a Recognizer Object, Starting a Recognition Session and Handling the Result**

```
private SpeechRecognizerUI speechInput = new SpeechRecognizerUI();

// Set text to display to the user when recognizing
speechInput.Settings.ExampleText = "Example: \"Buy roses\"";
speechInput.Settings.ListenText = "Say your memo";

// ...

// Private method to get a new memo
private async void GetNewMemoByVoice()
{
  await speechOutput.SpeakTextAsync("Say your memo"); // TTS prompt

  var recoResult =
    await speechInput.RecognizeWithUIAsync();
      // Uses default Dictation grammar

  Memo_TB.Text =
    recoResult.RecognitionResult.Text; // Do something with the result
}
```

the simple cases and show some of the richer functionality available for advanced scenarios.

Communicating to the User: Speech Synthesis API

Computer-generated speech output is variously called text to speech (TTS) or speech synthesis (though strictly speaking, TTS encompasses more than speech synthesis). Common uses include notification and confirmation, as mentioned earlier, but it's also essential to other use cases such as book readers or screen readers.

A Simple Example of Speech Synthesis In its simplest form, your app can translate a text string to spoken audio in just two lines of code. Here's an example using code extracted from the Magic Memo sample:

```
// Instantiate a speech synthesizer
private SpeechSynthesizer speechOutput = new SpeechSynthesizer();

// ...

// Private method to get a new memo
private async void GetNewMemoByVoice()
{
  await speechOutput.SpeakTextAsync("Say your memo");
  // Other code for capturing a new memo

}
```

When the user taps the mic button, she'll hear "Say your memo" spoken from the current audio device. In the following sections I'll expand on this example by adding code that accepts the user's input using speech recognition.

TTS Features for Advanced Scenarios Apps that rely heavily on speech output might have use cases that require changing volume, pitch or speaking rate in the course of speech output. To cover these advanced cases, there are two additional methods: SpeakSsmlAsync and SpeakSsmlFromUriAsync. These methods assume the input is in Speech Synthesis Markup Language (SSML) format, a World Wide Web Consortium (W3C) XML standard for embedding properties of the audio and the synthesizer engine into the text to be spoken. I haven't included sample code for SSML in this article or the Magic Memo code download, but you can find out more about SSML in the MSDN Library reference article at bit.ly/QwWLsu (or the W3C specification at bit.ly/V4DlgG).

The synthesizer class also has events for SpeakStarted and BookmarkReached, and there are overloads for each Speak method that take a generic state object as a second parameter to help you keep track of which instance of the Speak method generated a particular event. Using SSML and handling the events, your code can provide features such as highlighting spoken text or restarting a Speak call in the middle of a paragraph.

Speech Input: Speech Recognition API

The two broad classes of use cases for speech recognition in an app are text input and command and control. In the first use case, text input, the app simply captures the user's utterance as text; this is useful when the user could say almost anything, as in the "new memo" feature of the sample code.

In the second use case, command and control, the user manipulates the app by spoken utterance rather than by tapping buttons or sliding a finger across the face of the phone. This use case is especially useful in hands-free scenarios such as driving or cooking.

A Simple Example of Speech Recognition Before going into detail about the features of speech recognition in an app, let's take a look at the simplest case: text input in a few lines of code.

Figure 1 shows the GetNewMemoByVoice method shown earlier, but with lines added to initialize a recognizer object, start a recognition session and handle the result.

Of course, in real code it's never as simple as this, and if you look in the Magic Memo sample, you'll see a try/catch block and a check for successful recognition.

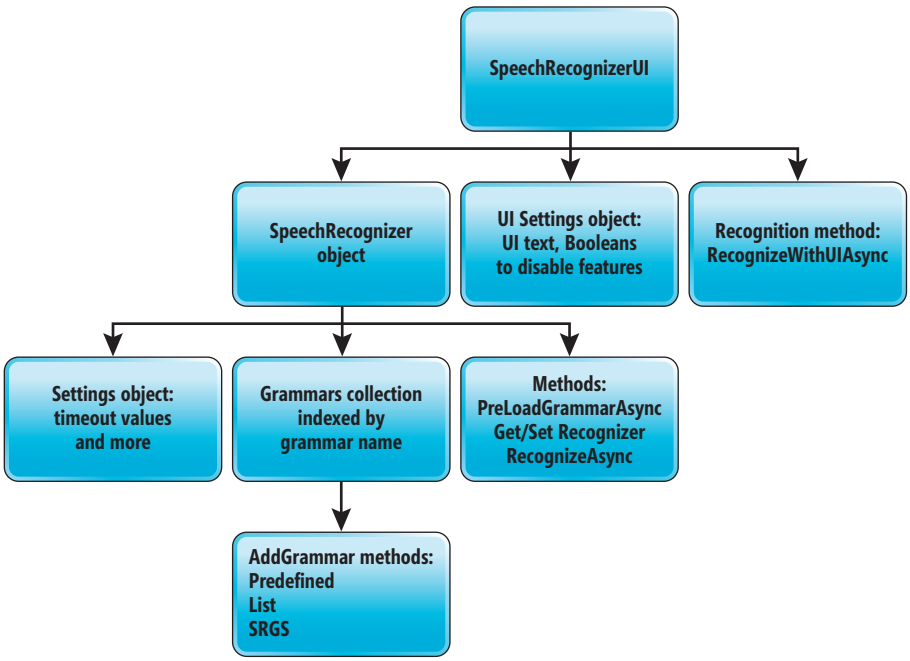


Figure 2 **Speech Recognition API Design Overview**

If you try this in the sample app by tapping the mic icon, you'll notice that after you've spoken your memo, a "thinking" screen appears, followed by a confirmation UI, after which the result is inserted in the memo text box. Behind the scenes a lot is going on, not the least of which is the use of a "grammar" on a remote server to recognize your speech. A grammar is essentially a set of rules specifying what lexical entries ("words") the engine needs to recognize and in what order. In the next sections I'll explore the speech recognition API and how it's used with recognition grammars.

Overview of the Speech Recognition API Before I get into the details of coding for speech recognition, let's take a high-level look at the classes in the API and their roles. **Figure 2** shows the basic layout of the API. The first thing you'll notice is that two boxes have SpeechRecognizer in the name.

Figure 3 **Processing the Result of a Recognition Session**

```
switch (result.RecognitionResult.Text.ToLower())
{
  case "cancel":
  // Cancel code
    break;

  case "save":
  // Save memo code

    break;

  case "quit":

    break;

  default:
    break;
}
```

If your app doesn't need to display a UI with speech recognition, or if you want to display your own custom UI, you should instantiate a copy of the SpeechRecognizer class shown in the middle-left of **Figure 2**. Think of this object as the base operational unit of speech recognition within this API. This is where the app adds any grammars it requires. After initialization, you call RecognizeAsync to do the actual recognition. Because SpeechRecognizer implements IAsyncOperation<SpeechRecognitionResult>, status and a result object are available in the Completed callback function. Thus, there are no separate events for recognition completed or rejected as in other managed speech APIs.

As the name implies, the top-level SpeechRecognizerUI class provides speech recognition with a default GUI that's consistent with the phone's global speech UI for feedback, disambiguation and confirmation. To maintain compatibility with the global speech UI and simplify coding, most apps should use this class rather than the non-UI class mentioned earlier. When you instantiate a SpeechRecognizerUI object, it comes with two important objects: a Settings object, where you set the UI text to display to the user; and a SpeechRecognizer object, where you can specify grammars as described in the following sections. Once initialized, you should call RecognizeWithUIAsync on the parent SpeechRecognizerUI object to launch a recognition session. If you use RecognizeAsync on the child SpeechRecognizer object, it will recognize as if the SpeechRecognizer object were being used standalone, that is, without a UI. Hereafter, the terms SpeechRecognizer and RecognizeAsync are understood to be generic references for the objects and methods with and without a UI, as appropriate.

Steps for Using Speech Recognition There are four basic steps for using speech recognition in a Windows Phone 8 app:

1. **Create grammars** to be used in the recognition process (not needed if using a predefined grammar).
2. **Initialize** the SpeechRecognizer object by setting properties and adding grammars as needed.
3. **Start the recognition session** by calling SpeechRecognizer.RecognizeAsync or SpeechRecognizerUI.RecognizeWithUIAsync.
4. **Process** the recognition result and take the appropriate action.

Figure 1 shows all of these steps except No. 1, Create grammars. The predefined Dictation grammar is the default grammar, so there's no need to create or add it to the Grammars collection.

The code to implement these steps largely depends on the type of grammar used in speech recognition. The next section describes the concept and use of speech recognition grammars in Windows Phone 8.

Figure 4 **Excerpts from ViewMemos.grxml SRGS Grammar**

```
<?xml version="1.0" encoding="utf-8"?>

<!DOCTYPE grammar PUBLIC "-//W3C//DTD GRAMMAR 1.0//EN"
  "http://www.w3.org/TR/speech-grammar/grammar.dtd">

<!-- the default grammar language is US English -->
<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
           http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0" tag-format="semantics/1.0"
           root="buttons">

  <rule id="buttons" scope="public">
    <one-of>
      <!--The 'process' semantic can be one of 'clear',
        'save', 'new', or 'select'-->
      <item>
        <!--Reference to the internal rule "scope" below-->
        Clear <ruleref uri="#scope" type="application/srgs+xml"/>
        <tag>out.process="clear";out.num = rules.latest();</tag>
      </item>
      <item>
        Save
        <item repeat="0-1">changes</item>
        <tag>out.process="save";</tag>
      </item>
      <item>
        Enter new
        <tag>out.process="new";</tag>
      </item>
      <item>
        Select
        <item repeat="0-1">memo</item> <!-- Optional words -->
        <item repeat="0-1">number</item>
        <!--Reference to the internal rule "number" below -->
        <ruleref uri="#number" type="application/srgs+xml"/>
        <tag>out.process="select";out.num =
          rules.latest();</tag>
      </item>
    </one-of>
  </rule>

  <rule id="scope" scope="private">
    <one-of> <!-- Can be "all", "selected" or a number from the
      'number' rule -->
      <item>
        all <tag>out.scope="all";</tag>
      </item>
      <item>
        selected <tag>out.scope="selected";</tag>
      </item>      <item>
        <item repeat="0-1">memo</item> <!-- Optional words -->
        <item repeat="0-1">number</item>
        <ruleref uri="#number" type="application/srgs+xml"/>
      </item>
    </one-of>
  </rule>

  <rule id="number" scope="public">
      <item>
        1
      </item>
    <!-- See ViewMemos.grxml for the remainder
      of the items in this block -->
  </rule>
</grammar>
```

BEST SELLER

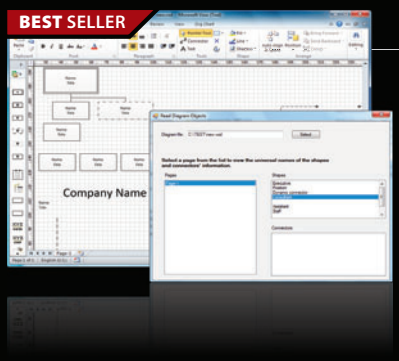
Help & Manual Professional | from $583.10

ec software

Easily create documentation for Windows, the Web and iPad.

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

BEST SELLER



Aspose.Diagram for .NET | from $587.02

ASPOSE
Your File Format Experts

Work with Visio files from within your own applications.

- Work with VSD, VSS, VST, VSX, VTX, VDW and VDX files on C#, VB.NET, ASP.NET Web applications, Web services, Mono and Windows applications
- Export to popular formats including PDF, XPS, BMP, JPEG, PNG, TIFF, SVG and EMF
- Easy to deploy - no external dependencies aside from the .NET Framework
- Access Visio objects like Document, Page, Master, Shape, StyleSheet and Connect etc.

BEST SELLER

LEADTOOLS Medical Imaging SDKs V17.5 | from $4,495.50

LEAD TECHNOLOGIES

Add powerful medical imaging support to desktop, tablet, mobile & web applications.

- Zero footprint HTML5 / JavaScript viewer for any desktop, tablet or mobile device
- 2D / 3D Viewers and 3D Volume Rendering
- Comprehensive DICOM Data Set and metadata
- High level PACS SDK with an OEM-Ready PACS Storage Server & Viewer application
- Advanced 8-16 bit grayscale image processing including Window Level and Signed Data

BEST SELLER

ComponentOne Studio Enterprise | from $1,315.60

ComponentOne
a division of GrapeCity

.NET Tools for the Smart Developer: Windows, Web, and XAML.

- Hundreds of UI controls for all .NET platforms, including grids, charts, reports and schedulers
- Visual Studio 2012 and Windows 8 Support
- Live demos and samples with full source code
- Royalty-free deployment and distribution
- Free, fully-functional trial download available

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

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

European Headquarters
ComponentSource
30 Greyfriars Road
Reading
Berkshire
RG1 1PE
United Kingdom

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

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

GSA Schedule Contract GS-35F-0188R

Introduction to Speech Recognition Grammars

Modern speech recognition engines all use grammars to restrain the set of phrases through which the recognition engine must search (hereafter called the "search space") to find a match to the user's utterance, and thus improve recognition accuracy. Grammar rules may allow recognition of phrases as simple as a list of numbers or as complex as general conversational text.

In the Windows Phone 8 speech API you can specify a grammar in three ways, as described in the following sections. For each case, you add the grammar to a collection of grammars on the SpeechRecognizer object.

Simple List Grammar The easiest way to specify a custom grammar for an app is to provide a list of all the phrases for which the recognizer should listen in a simple string array. These list grammars are handled by the on-device speech recognition engine. The code to create and add a list grammar can be as simple as the following for a static list of button names to recognize against:

```
commandReco.Recognizer.Grammars.AddGrammarFromList(
  "mainPageCommands", new string[] { "cancel", "save", "quit" });
```

The Magic Memo sample does something a little more sophisticated: It builds up the list of phrases by finding the Content attribute of all the button controls on the page and adding the content text to a string list. See the InitCommandGrammar method in MainPage.xaml.cs for details.

To process the result of a recognition session using a list grammar, you read the Text property on SpeechRecognitionUIResult (or SpeechRecognitionResult if using the version without a UI). You could do this, for example, in a switch statement, as shown in **Figure 3**.

A more detailed example is found in the CommandCompleted callback in MainPage.xaml.cs.

Predefined Grammar The Speech API on Windows Phone 8 provides two predefined grammars: Dictation and WebSearch. Dictation is also called Short Message Dictation and employs the same grammar as used in the built-in Texting app. In contrast, WebSearch is optimized to the phrases used to search online. The built-in Find/Search command uses the same WebSearch grammar.

The search space for both predefined grammars is vast, requiring the processing power available through remote speech recognition using the Microsoft speech Web service. In general these grammars aren't well suited to command and control because of the possibility of misrecognition and the wide range of possible results.

A major advantage of predefined grammars is that they're easy to implement in an app. For example, to use the WebSearch grammar rather than the default Dictation grammar in **Figure 1**, you simply add this line before the call to RecognizeWithUIAsync:

```
speechInput.Recognizer.Grammars.AddGrammarFromPredefinedType(
  "webSearch", SpeechPredefinedGrammar.WebSearch);
```

You process the recognition result from a predefined grammar by accessing the result Text property, as shown in **Figure 1**.

Grammars in Speech Recognition Grammar Specification Format The Speech Recognition Grammar Specification (SRGS) is a W3C standard in XML format. For details about the format and usage, see the MSDN Library article, "SRGS Grammar XML Reference," at bit.ly/SYnAu5; the W3C specification at bit.ly/V4DNeS; or any number of tutorial Web pages that you'll find by searching online for "SRGS grammar." SRGS grammars offer rich functionality such as the ability to specify optional items and to repeat items, rules, rule references, special rules and semantics—at the expense of extra effort to author, test and debug the grammar. In Windows Phone 8, SRGS grammars are used only in the local recognition engine on the phone, that is, not in the remote service.

To add an SRGS grammar, you reference the URI of the grammar file in the app's install path, as follows:

```
commandReco.Recognizer.Grammars.AddGrammarFromUri(
  "srgsCommands", new Uri("ms-appx:///ViewMemos.grxml"));
```

One major advantage of SRGS grammars is that they allow you to specify semantic values to simplify the processing of a wide range of user responses without accessing the recognized utterance (which is available on the RecognitionResult.Text property, as always).

SRGS semantics are objects (which in practice are often strings) that you assign to variables in your SRGS grammar using a <tag> element and a subset of ECMAScript. They have two advantages over using the recognized text directly:

1. **Simplified processing:** You can determine the user's intent without parsing the recognized text, which might take on

Figure 5 **Handling a Recognition Result Using Semantic Properties**

```
// micImage Tap handler, excerpted from ViewMemos.xaml.cs
private async void micImage_Tap(object sender, GestureEventArgs e)
{
  var commandResult = await commandReco.RecognizeWithUIAsync();

  if (commandResult.ResultStatus ==
    SpeechRecognitionUIStatus.Succeeded)
  {
    var commandSemantics = commandResult.RecognitionResult.Semantics;

    SemanticProperty process = null;

    if (commandSemantics.TryGetValue("process", out process))
    {

      // In general a semantic property can be any object,
      // but in this case it's a string
      switch (process.Value as string)
      {
        // For this grammar the "process" semantic more or less
        // corresponds to a button on the page
        case "select":
        // Command was "Select memo number 'num'"

          break;

        case "clear":
        // Command was "Clear memo number 'num,'" "Clear all"
        // or "Clear Selected"

          break;

        case "save":
        // Command was "Save" or "Save Changes"

          break;

        case "new":
        // Command was "Enter new"

          break;

        default:
          break;
      }
    }
  }
}
```

multiple forms for the same meaning. For example, using semantics, you can map all utterances that mean affirmative—"yes," "yup," "affirmative," "OK" or "ya"—to the single semantic value "yes."

2. **Ease of localization:** You can use the same codebehind to process utterances in any supported spoken language if you use a uniform set of semantic values across all languages.

To illustrate these concepts, the Magic Memo sample uses a simple grammar ViewMemos.grxml for controlling the ViewMemos.xaml page; excerpts from that grammar file with the semantic tags are shown in **Figure 4**. The function micImage_Tap in ViewMemos.xaml.cs (excerpted in **Figure 5**) demonstrates the use of semantic values in mapping the user's utterance to an action.

This sample just scratches the surface of what's possible with semantics. To explore more, start with the MSDN Library article, "Using the tag Element," at bit.ly/PA80Wp. The W3C standard for semantics is at bit.ly/RyqJxc.

You can try out this grammar in the Magic Memo sample by navigating to the ViewMemos page and tapping the mic icon. The file ViewMemos.xaml.cs has the codebehind, including code under a #define section that you can activate (using #define SemanticsDebug) to display and debug the semantic values returned on the recognition result.

Using Multiple Grammars on the Same Recognizer Object

A natural question to ask at this point is whether you can use more than one grammar on a SpeechRecognizer object. The answer is yes, with some restrictions. Here are some guidelines and coding techniques for using multiple grammars:

1. If you add a predefined grammar, you can't add any other grammars. Also, you can't disable a predefined grammar; it's the one and only grammar associated with that recognizer object for its lifetime.
2. You can add multiple custom grammars (list grammars and SRGS grammars) to a single recognizer object and enable or disable the grammars as needed for different scenarios in your app:
   a. To access a specific grammar, use the grammar name (the string parameter passed in the call to the AddGrammar method) as a key on the Grammars collection.
   b. To enable or disable a particular grammar, set its Enabled Boolean to true or false. For example, the following will disable the grammar named "buttonNames":

```
myRecognizer.
Grammars["buttonNames"].Enabled =
false;
```

3. When you call any of the AddGrammar methods, the grammar is put in a queue to await processing but isn't parsed or loaded. The grammar is compiled and loaded on the first call to RecognizeAsync or on an optional call to PreLoadGrammarsAsync. Calling this latter method before actual use can reduce the latency in returning a result from RecognizeAsync and is therefore recommended for most use cases.

The Next 'Killer App'

The speech features for apps on Windows Phone 8 represent, among all smartphone offerings, the first fully functional developer platform for speech featuring both on-device and remote recognition services. Using voice commands and in-app dialog, you can open up your app to many compelling scenarios that will delight your users. With these speech features, your app could catch the buzz and be the next "killer app" in the marketplace. ■

F Avery Bishop *has been working in software development for more than 20 years, with 12 years spent at Microsoft, where he's a program manager for the speech platform. He has published numerous articles on natural language support in applications including topics such as complex script support, multilingual applications and speech recognition.*

Thanks to the following technical experts for reviewing this article:
Eduardo Billo, Rob Chambers, Gabriel Ghizila, Michael Kim and Brian Mouncer

Designing Accessibility with HTML5

Rajesh Lal

If you're truly interested in reaching a broad audience, you'll want to design your Web site for accessibility. Accessibility is about making Web pages easier to access, easier to use and available to everyone. In general, using the latest technologies makes accessibility easier to accomplish. Today, that means using HTML5.

To be accessible, your content needs to be available on a broad range of devices, such as ordinary computers using a keyboard or mouse, screen readers, audio browsers, devices with limited bandwidth, old browsers and computers, and mobile phones and touch devices. Moreover, it should be reachable by the widest variety of people, including those with disabilities and senior citizens, as well as people with low literacy levels or temporary illness, or who prefer using only a keyboard or mouse.

This article discusses:

- The W3C POUR model for accessibility
- Progressive enhancement and ARIA
- Creating an accessible Web site with HTML5
- Accessibility support in Visual Studio 2012

Technologies discussed:

HTML5, WAI-ARIA, Visual Studio 2012

Code download available at:

archive.msdn.microsoft.com/mag201212HTML5

The four key areas disability accessibility seeks to address are:

- Hearing
- Mobility
- Cognitive
- Visual

Hearing issues mean a user may not be able to hear any sound on the Web site. The solution is to make the content *perceivable* by using a text alternative for all non-text content, such as subtitles and closed captions. Include transcribed speech and sign languages, if possible.

Mobility problems in this case mean the inability to use the mouse or keyboard. The solution for mobility on the Web is to make the content *operable*; that is, to make all functionality accessible from the keyboard alone, as well as with joysticks, voice recognition and audio feedback, when possible. Allow navigation with proper use of headings and anchors and give users the ability to stop time-based content. Don't allow any auto-refresh on the page.

Cognitive difficulties impact the content itself, for example with the size of text and images or with color contrast. Flashy graphics and font types can also cause problems for some users. The solution is to make the content *understandable*. Use easy-to-read sans serif fonts and allow font resizing. Use high color contrast between foreground and background. Avoid auto-refresh, flickering images and auto play of media and animation. Use multiple visual cues and standard icons to make the content easy to grasp.

Visual problems can range from an inability to distinguish color to no ability to see the content at all. The solution for such issues is to make the content *robust* so that it can be reliably interpreted by user agents, and easily accessed with screen readers. Use semantic HTML and follow standards. Use syntactically correct HTML and validate your page. Use *lang* attribute and *abbr* tags wherever applicable.

In short, to be accessible, content for the Web needs to be made perceivable, operable, understandable and robust. Together, these attributes comprise the World Wide Web Consortium (W3C) POUR model, which mandates that the information and UI elements being presented to users must be perceivable to their senses; that there must be a way for them to operate the UI; that they must be able to understand the information and how to use the interface elements; and that the content be robust enough so they can access it using a variety of user agents, including assistive technologies (ATs).

Now that you understand the fundamentals of accessibility, let's take a look at two very important concepts related to accessible Web design: progressive enhancement and accessible rich Internet applications (ARIAs).

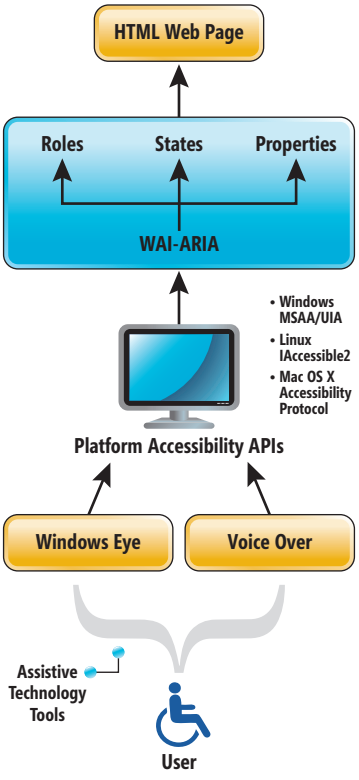


Figure 1 **How a User Accesses a Web Page Using Assistive Technology**

Progressive Enhancement and ARIA

Progressive enhancement is an approach to Web design that promotes accessibility using semantic HTML, stylesheets and scripting. The idea is to create a Web site where basic content is available to everyone while more advanced content and functionality are accessible to those with more capability, more bandwidth or more advanced tools. When you create a site, you concentrate first on displaying the content in the simplest manner. You design your page using semantically structured HTML. All presentation elements that modify the visual content (such as bold or italics) go in an external stylesheet.

Semantic HTML means the HTML tags in a page should describe the content in a way that has to do with its meaning rather than its presentation. Any information about the decoration of the content should go in a CSS file, while the logic and the client-side behavior of the Web page should be added via externally linked JavaScript after the page is loaded and the stylesheet parsed and applied. Progressive enhancement ensures that if there's an error in the JavaScript file, the page still loads with proper styles. And if the CSS file is ignored (for example, by screen readers), the HTML page still has all the content.

All modern OSes have their own accessibility APIs, each of which is a set of open methods and interfaces exposed by the browser for reading and parsing text. The Microsoft version is Microsoft Active Accessibility (MSAA), a part of UI Automation (UIA) for Windows; Linux has IAccessible2; and Apple relies on the Mac OS X Accessibility Protocol—but they all follow the ARIA standard as defined by the W3C (bit.ly/OID4IX). **Figure 1** shows how a user might interact with an accessible Web page using an AT device such as a screen reader. Such devices use the accessibility APIs to access Web pages.

ARIA is part of the W3C Web Accessibility Initiative (WAI), and it defines a way to make Web content and Web applications more accessible. ARIA is used to improve the accessibility of dynamic content and advanced UI controls developed with HTML, CSS, JavaScript, AJAX and related technologies. ARIA is now officially a part of the HTML5 specification and is also embedded in popular JavaScript libraries such as JQuery, Dojo and YUI. See bit.ly/b89BEJ for more information.

ARIA uses a set of roles, states and properties to expose a Web page to the accessibility APIs. These roles, states and properties are assigned on a page's elements, which are exposed to the ATs. Most current AT tools—including JAWS, NVDA and VoiceOver—support ARIA. Let's take a take a closer look at ARIA.

ARIA Roles

Roles indicate the type of element in a meaningful way. Suppose a screen reader comes across an HTML element on a page that includes role=navigation. The screen reader will know this HTML element is

Figure 2 **ARIA Role Values**

Landmark Roles	Structural Roles		Widget Roles		
			Standalone Widgets		Composite Widgets
application	article	region	alert	progressbar	combobox
banner	columnheader	row	alertdialog	radio	grid
complementary	definition	rowheader	button	scrollbar	listbox
contentinfo	directory	separator	checkbox	slider	menu
form	document	toolbar	dialog	spinbutton	menubar
main	group		gridcell	status	radiogroup
navigation	heading		link	tab	tablist
search	img		log	tabpanel	tree
	list		marquee	textbox	treegrid
	listitem		menuitem	timer	
	math		menuitemcheckbox	tooltip	
	note		menuitemradio	treeitem	
	presentation		option		

Figure 3 **ARIA States and Properties**

Attribute Type	Global	Widget
ARIA States	aria-busy aria-disabled aria-grabbed aria-hidden aria-invalid	listitem math note presentation region row rowheader separator toolbar
ARIA Properties	aria-atomic aria-controls aria-describedby aria-dropeffect aria-flowto aria-haspopup aria-label aria-labelledby aria-live aria-owns aria-relevant	aria-autocomplete aria-haspopup aria-label aria-level aria-multiline aria-multiselectable aria-orientation aria-readonly aria-required aria-sort aria-valuemax aria-valuemin aria-valuenow aria-valuetext

for navigation, and the user will be able to access navigation directly instead of tabbing through all the links.

ARIA role attributes are applied to HTML elements like this:

```
<div role="XXX"> </div>
```

> Accessibility is about making Web pages easier to access, easier to use and available to everyone.

Here "XXX" is a value that depends on the type of the HTML element and its role on the page. It can take a number of values—such as a form, navigation, search or article—based on the content it represents. There are three types of roles:

- Landmark roles act as navigational landmarks.
- Structural roles define the document's structure and help organize content.
- Widget roles consist of standalone UI widgets as well as composite widgets that are containers of two or more standalone widgets.

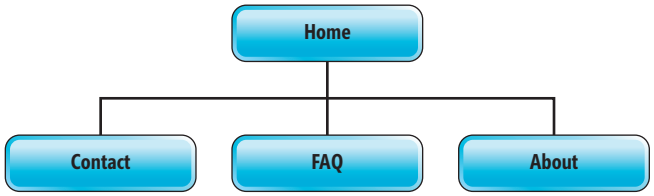


Figure 4 **Sitemap for Web Site Example**

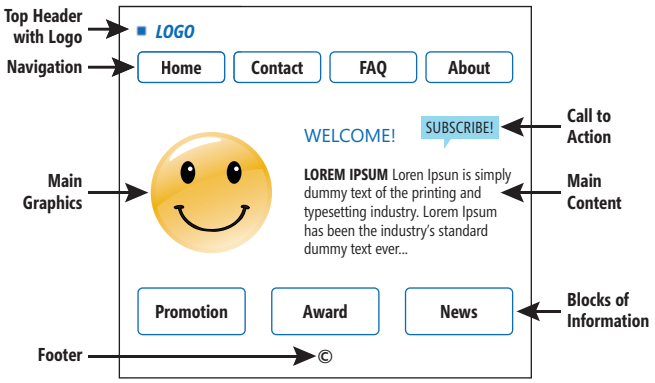


Figure 5 **Standard Layout for a Homepage**

Figure 2 shows all the role values available in ARIA. There are eight landmark roles, 18 structural roles, 25 standalone interface widget roles and nine composite UI widget roles in ARIA. You'll find more information at bit.ly/SOHUvi.

Unlike roles, ARIA states and properties are attributes that can be set for each HTML element.

ARIA States

An ARIA state is a dynamic property of an HTML element that represents data associated with the object but doesn't affect the essential nature of the element. There are two types of ARIA states—global and widget—as shown in **Figure 3**. Global states can be

Figure 6 **HTML5 Homepage**

```
<!doctype html>
<html lang="en">
<head><title>HTML5 Home Page</title></head>
<body>
  <header><!-- header -->
  <a href="/"><img src="images/logo.png"></a>
  </header>
<nav><!-- navigation -->
<ul>
  <li><a href="/">Home</a></li>
  <li><a href="/contact">Contact</a></li>
  <li><a href="/faq">FAQ</a></li>
  <li><a href="/about">About</a></li>
</ul>
</nav>
<div><!-- main content -->
<section>
  <figure><img src="images/maingraphics.png">
  <figcaption>Welcome image. More help on image<a href="#/">Help</a></figcaption>
  </figure>
</section>
<section>
  <h2><a href="Action" target="_blank">Subscribe</a></h2>
  <article>
  <h2>Welcome!</h2>
  <p><strong>Lorem Ipsum </strong>is simply dummy text of the printing and … </p>
  </article>
</section>
</div>
<aside><!-- info blocks -->
  <h4>Promotion</h4><ul><li>items</li></ul>
  <h4>Awards</h4><ul><li>items</li></ul>
  <h4>News</h4><ul><li>items</li></ul>
</aside>
<footer><!-- footer -->
  <div>Copyright © 2012</div>
  <div><a href="">Privacy Policy</div>
</footer>
</body>
</html>
```

Figure 7 **Adding Roles**

```
<!doctype html>
<html lang="en">
<head><title> Accessible HTML5 Home Page</title></head>
<body>
  <header role="banner"><!-- header -->
  <a href="/"><img src="images/logo.png"></a>
  </header>
<nav role="navigation"><!-- navigation -->
<ul>
  <li><a href="/">Home</a></li>
  <li><a href="/contact">Contact</a></li>
  <li><a href="/faq">FAQ</a></li>
  <li><a href="/about">About</a></li>
 </ul>
</nav>
<div id="maincontent" role="main"><!-- main content -->
<section>
 <figure><img src="images/maingraphics.png">
 <figcaption>Welcome image. More help on image<a href="#/">Help</a></figcaption>
 </figure>
</section>
<section role="region">
  <h2><a href="Action" target="_blank">Subscribe</a></h2>
  <article role="article">
  <h2>Welcome!</h2>
  <p><strong>Lorem Ipsum </strong>is simply dummy text of the printing and … </p>
  </article>
</section>
</div>
<aside role="complementary"><!-- info blocks -->
  <h4>Promotion</h4><ul><li>items</li></ul>
  <h4>Awards</h4><ul><li>items</li></ul>
  <h4>News</h4><ul><li>items</li></ul>
</aside>
<footer role="contentinfo"><!-- footer -->
  <div>Copyright © 2012</div>
  <div><a href="">Privacy Policy</div>
</footer>
</body>
</html>
```

applied to any element regardless of whether a role has been applied to the element. Widget states are attributes of UI widgets that require user interaction.

The following shows the attribute aria-hidden:

```
<div aria-hidden="true">
  <p>Paragraph text here </p>
</div>
```

This code will hide the paragraph from a screen reader.

ARIA Properties

ARIA properties are similar to ARIA states but are relatively static on the page and act as additional properties of the HTML element.

Figure 8 **Form Input Types and Attributes**

Input Type	input type=datetime input type=datetime-local input type=date input type=month input type=time input type=week input type=number	input type=range input type=email input type=url input type=search input type=tel input type=color
Attributes	autocomplete autofocus form formaction formenctype formmethod formnovalidate	formtarget list multiple pattern placeholder required step

Widget properties are analogous to widget states but the value doesn't change within the scope of the page. There are 11 global properties and 14 widget properties (see **Figure 3**).

Here's an example of the widget property aria-required:

```
<label for="username">User name:</label>
<input id="username" type="text" aria-required="true">
```

This makes a form's input field required.

Figure 3 summarizes all of the ARIA states and properties. Visit bit.ly/OlbLeh for more information.

Now that you have some familiarity with ARIA and its roles, states and properties, you'll be able to use it to create a progressively enhanced accessible Web site.

Creating an Accessible Web Site

A typical Web site contains a number of components. Let's take a look at how to create each of the following, keeping accessibility in mind and using HTML5 and ARIA:

1. Homepage
   - Header area with logo
   - Navigation menu
   - Main graphics
   - Main content
   - Blocks of information
2. Contact form
3. FAQ page
4. About page with video

Figure 4 shows the basic structure of the site I'll create.

Figure 5 shows a typical homepage layout for a product- or service-based site. To create it, I'll first use HTML5 with progressive enhancement and then make it accessible to AT devices.

Figure 9 **Creating an HTML5 Contact Form**

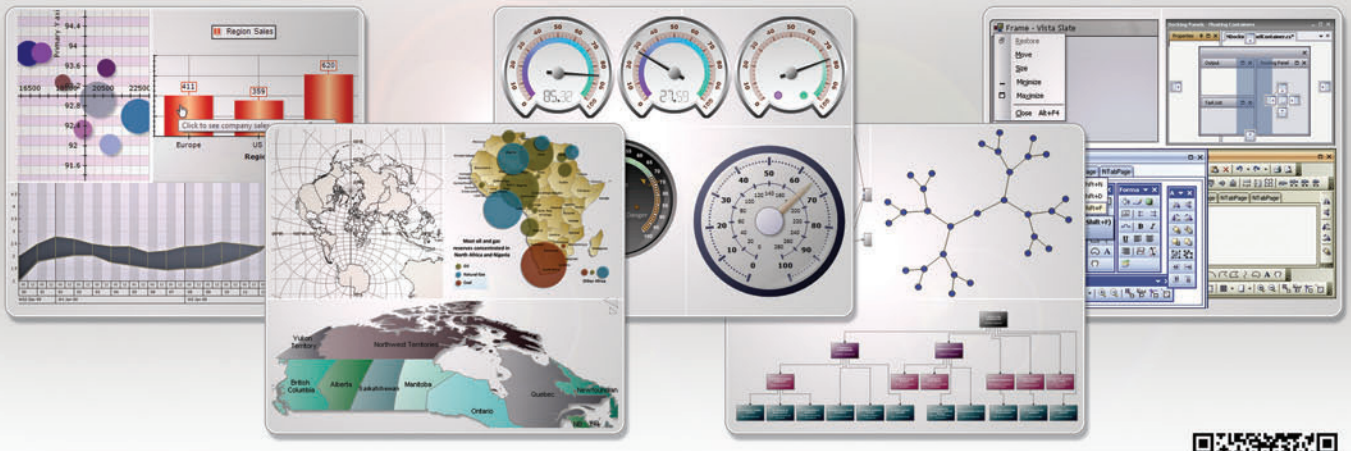
```
<div id="contact" role="main"><!-- main content -->
  <!-- content -->
    <section id="content">
      <article>
        <h2>Contact <span>Form</span></h2>
        <form id="contacts-form" action="" method="post">
          <fieldset>
            <div class="field">
              <label for="name" >Name </label>
              <input id="name" placeholder="John Smith" autofocus required
                aria-required="true" type="text" value="" />
            </div>
            <div class="field">
              <label for="email">E-mail</label>
              <input id="email" placeholder="john@msn.com" type="email" required
                aria-required="true" value=""/>
            </div>
            <div class="field">
              <label for="website">Website</label>
              <input id="website" placeholder="http://website.com"
                type="url" />
            </div>
            <div class="field">
              <label for="message">Message</label>
              <textarea id="message"
                placeholder="Write your message Here!" required
                aria-required="true" ></textarea>
            </div>
            <div><a href="#" onclick="submit()">Send Your Message!</a></div>
          </fieldset>
        </form>
      </article>
    </section>
</div>
```

NEVRON
visualize your success

NEVRON
GAUGE for .NET, SSRS, SharePoint

NEVRON
CHART for .NET, SSRS, SharePoint

NEVRON
USER INTERFACE for .NET



NEVRON
MAP for .NET

NEVRON
DIAGRAM for .NET

2012vol.1 is here

Features new ThinWeb controls with JQuery and ASP.NET MVC integration, SQL Server and SharePoint "Expressions Everywhere" and more.

Nevron components integrate seamlessly in **Web** and **Desktop .NET** applications, **SQL Server Reporting Services 2005/2008** reports and **SharePoint 2007/2010** portals and deliver an unmatched set of enterprise-grade features. That is why Nevron is the trusted vendor by many Fortune 500 companies for their most demanding data visualization needs.

Download your free evaluation from

www.nevron.com

DEVELOPERS

.NET

IT PROFFESIONALS

SharePoint

SSRS

Microsoft .NET ASP.net AJAX Microsoft SharePoint 2007/2010 Microsoft SQL Server 2005/2008 Microsoft Visual Studio PARTNER

Figure 10 **The Contact Form Page in the Browser**

As you can see, I've identified a number of elements in the page: header, navigation, call to action, main graphics, welcome message with brief content, blocks of information and footer.

Following progressive enhancement principles, I create a sequential HTML5 page to accommodate these elements, using the elements <header>, <nav>, <figure>, <article>, <section>, <aside> and <footer>, as shown in **Figure 6**.

This code is supported in most current browsers, and any HTML5 element not supported defaults to a element. For example, if the <header> element isn't supported, the browser would substitute a , like so:

```
<header><!-- header -->
<a href="/"><img src="images/logo.png"></a>
</header>

<div><!-- header -->
<a href="/"><img src="images/logo.png"></a>
</div>
```

To allow AT tools to recognize the navigational landmarks and structural parts of the document, I add the following roles to each element, as shown in **Figure 7**:

- header role=banner
- nav role=navigation
- maincontent role=main
- section role=region
- article role=article
- aside role=complementary
- footer role=contentinfo

To apply styles for all browsers, the first step is to make all of the HTML5 elements block-level elements in the stylesheet, like this:

```
<style>
header,footer,nav,article,aside,section,figure,figcaption{display:block;}
</style>
```

I include the ARIA roles so the styles are applied for each particular element:

```
<style>
  header[role="banner"]{/* Styles for banner */}
  header{/* Styles for other headers */}
  #maincontent[role="main"]{ /* Styles for main content */}
  nav[role="navigation"]{/* Styles for navigation */}
  section[role="group"]{/* Styles for section */}
  article[role="article"]{/* Styles for article */}
  aside[role="complementary"]{/* Styles for info blocks */}
  footer[role="contentinfo"]{/* Styles */}
</style>
```

Figure 11 **Markup for a FAQ Page**

```
<h1>FAQ</h1>
  <h2>List of frequently asked questions</h2>
    <ul>
    <li><a href="#q1">Accessible Text</a></li>
    <li><a href="#q2">Accessible Tables</a></li>
    <li><a href="#q3">Accessible Links</a></li>
    <li><a href="#q4">Accessible Images</a></li>
    <li><a href="#q5">Accessible Titles</a></li>
    </ul>
  <h2 id="q1">Accessible Text</h2>
    <h3>Semantic HTML</h3>
    <h3>Proper hierarchy</h3>
    <h3>Localized content</h3>
    <h3>Acronym</h3>
    <h3>Font-size</h3>
    <h3>Color</h3>
  <h2 id="q2">Accessible Table</h2>
  <h2 id="q3">Accessible Links</h2>
  <h2 id="q4">Accessible Images</h2>
  <h2 id="q5">Accessible Titles</h2>
```

Because the HTML page is parsed sequentially, the best place to put the JavaScript file is at the bottom of the page, after the footer. This lets the site be completely independent of JavaScript—the JavaScript function is instantiated only after the document is ready and fully loaded. The following code shows the script file inserted in my example:

```
  <footer role="contentinfo"><!-- ><!-- footer -->
    <div>Copyright © 2012</div>
    <div><a href="">Privacy Policy</div>
  </footer>
    <script type="text/javascript" src="jquery.min.js"></script>
    <script type="text/javascript" src="main.js"></script>
  </body>
  </html>
```

Creating an Accessible Contact Form

Forms are an integral part of Web-based interaction, and HTML5 has a number of new input types and attributes that aid accessibility. **Figure 8** lists the ones related to HTML5 forms.

Figure 12 **An Accessible Table**

```
<h4>Table with Caption, Summary and Details</h4>
<table>
  <caption>
    <strong>Lorem Ipsum.</strong>
    <details>
      <summary>Help</summary>
      <p><strong>Lorem Ipsum </strong>is simply dummy text of the printing and </p>
     </details>
  </caption>
  <thead>
    <tr>
      <th>Table header column 1</th>
      <th>Table header column 2</th>
      <th>Table header column 3</th>
    </tr>
  </thead>
  <tfoot>
    <tr>
      <td>Table footer column 1</td>
      <td>Table footer column 2</td>
      <td>Table footer column 3</td>
    </tr>
  </tfoot>
  <tbody>
    <tr>
      <td>Table data column 1</td>
      <td>Table data column 2</td>
      <td>Table data column 3</td>
    </tr>
  </tbody>
</table>
```

Does your Team do more than just track bugs?

Free Trial and Single User FreePack™ available at www.alexcorp.com

Alexsys Team® does! Alexsys Team 2 is a multi-user Team management system that provides a powerful yet easy way to manage all the members of your team and their tasks - including defect tracking. Use Team right out of the box or tailor it to your needs.

Track all your project tasks in one database so you can work together to get projects done.

- Quality Control / Compliance Tracking
- Project Management
- End User Accessible Service Desk Portal
- Bugs and Features
- Action Items
- Sales and Marketing
- Help Desk

Native Smart Card Login Support including Government and DOD

New in Team 2.11

- Full Windows 7 Support
- Windows Single Sign-on
- System Audit Log
- Trend Analysis
- Alternate Display Fields for Data Normalization
- Lookup Table Filters
- XML Export
- Network Optimized for Enterprise Deployment

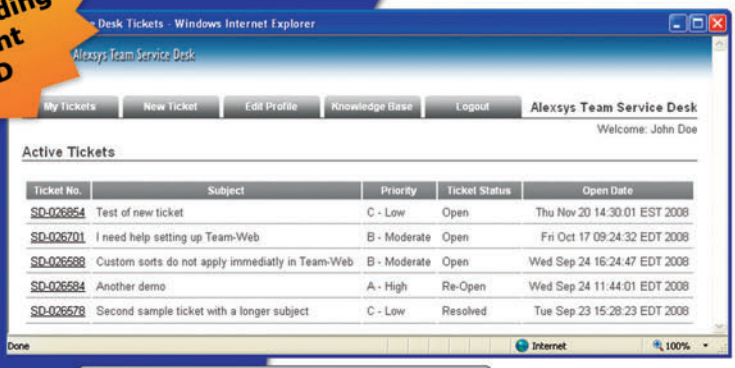
Service Desk Features

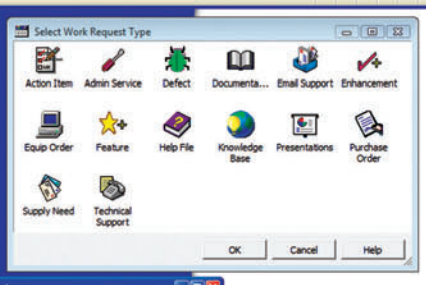
- Fully Secure
- Unlimited Users Self Registered or Active Directory
- Integrated into Your Web Site
- Fast/AJAX Dynamic Content
- Unlimited Service Desks
- Visual Service Desk Builder

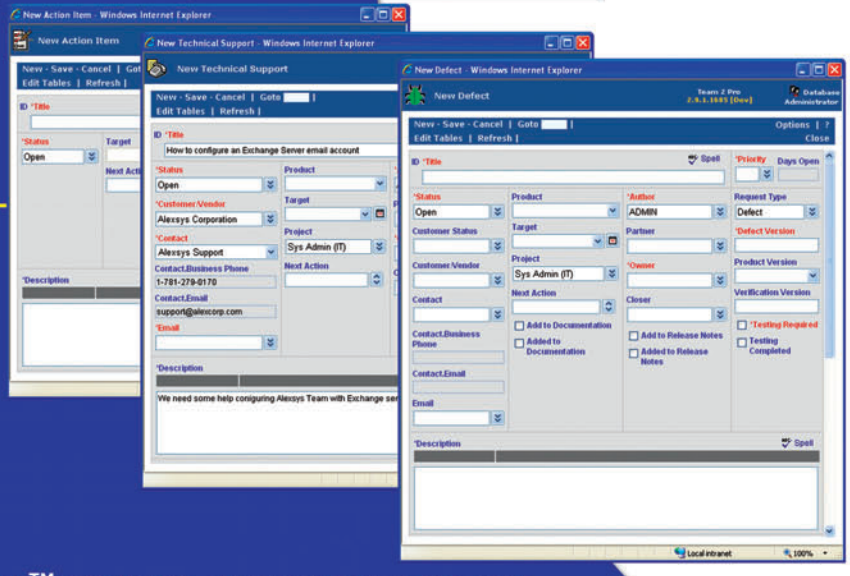
Team 2 Features

- Windows and Web Clients
- Multiple Work Request Forms
- Customizable Database
- Point and Click Workflows
- Role Based Security
- Clear Text Database
- Project Trees
- Time Recording
- Notifications and Escalations
- Outlook Integration

ZDNet Editors' Pick







Free Trial and Single User FreePack™ available at www.alexcorp.com.
FreePack™ includes a free single user Team Pro and Team-Web license.
Need more help? Give us a call at 1-888-880-ALEX (2539).

Team 2 works with its own standard database, while Team Pro works with Microsoft SQL, MySQL, and Oracle Servers.
Team 2 works with Windows 7/2008/2003/Vista/XP.
Team-Web works with Internet Explorer, Firefox, Netscape, Safari, and Chrome.

For accessibility, a form should be restricted to a single purpose. A contact page should contain only the contact form and no other distraction. This makes it much easier for people using AT devices.

It's also important to use the proper input type. This improves the UX for devices that support that attribute. For example, input type=number can show a numeric keypad for mobile devices while input type=url displays a special ".com" button in the virtual keyboard of many smartphones.

You use the for attribute in a label along with the id attribute in the input element, as follows:

```
<label for="useremail">Your E-mail:</label>
<input id="useremail" name="useremail" type="email" value=""/>
```

This maps the label to the input element in the assistive device. You could also do this in a more descriptive way using the aria-describedby attribute. For example, if you have some help text for each input field, you can wire it up with the input text:

```
<label for="useremail">Your E-mail:</label>
<input id="useremail" type="email" value="" aria-describedby="helpemail"/>
<p id="helpemail">Your email address will be used for further communication</p>
```

The next step is adding the placeholder and required HTML5 attributes (with aria-required="true"). The placeholder attribute lets you show what valid input looks like and the required attribute makes the input box a required field:

```
<label for="useremail">Your E-mail:</label>
<input id="useremail" type="email" placeholder="john@msn.com" required
aria-required="true" value="" aria-describedby="helpemail"/>
<p id="helpemail">Your email address will be used for further communication</p>
```

Note that placeholder is not a label. And keep in mind that if you use an asterisk with the text to indicate a required field, the asterisk is read by screen readers with every field, providing a poor UX for the visually impaired. Instead, use the aria-required field attribute, which tells the AT device the field is required, and use a background color or image rather than the asterisk to indicate that to the user.

You can also add the autofocus attribute, which helpfully sets the focus to the first element of the form.

Figure 9 shows code that creates an accessible HTML5 contact form, and **Figure 10** displays the contact form.

> To be accessible, your content needs to be available on a broad range of devices.

Updates can be difficult for people with disabilities, but live regions can make assistive devices aware of updates when you use the aria-live attribute along with the role attributes of status, log and alert:

- aria-live="off": updates are not announced (the region is not live)
- aria-live="polite": updates are announced when the user is idle
- aria-live="assertive": higher priority, but updates aren't necessarily announced immediately
- role="log", role="status" and role="alert" for different types of messages

Here's a simple way to integrate this into the HTML code:

```
<div id="liveregion" role="log" aria-live="polite">
```

Now let's look at a FAQ page with accessible content.

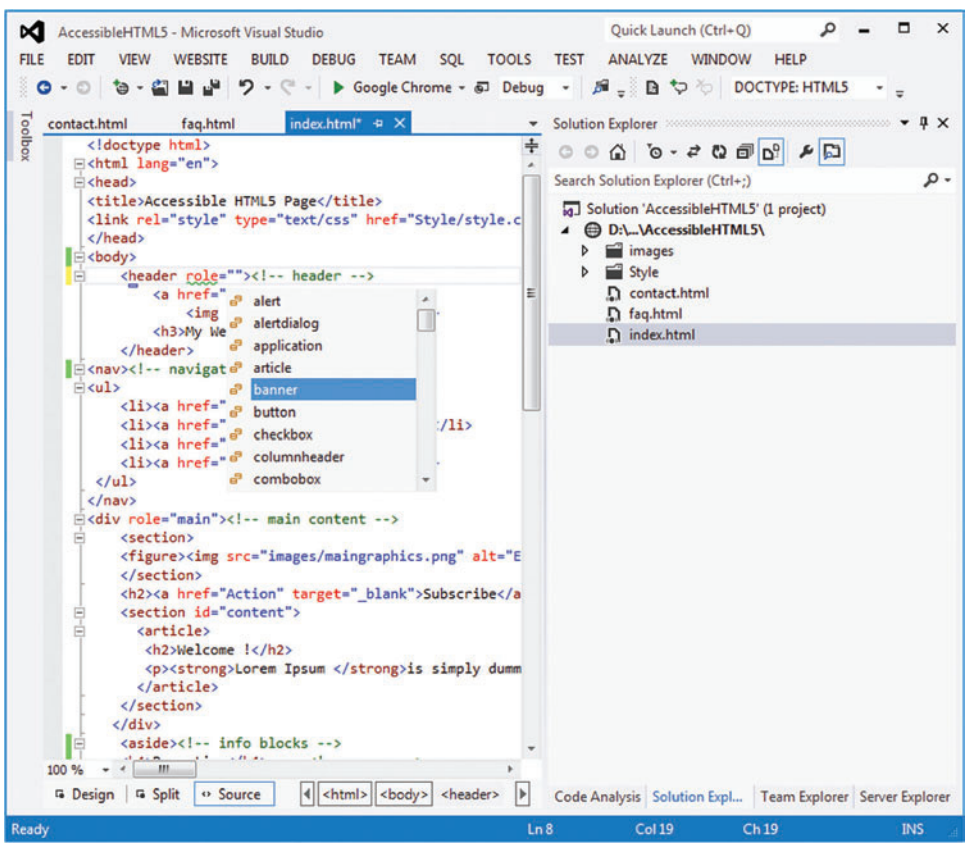


Figure 13 **IntelliSense Support for ARIA Roles in Visual Studio 2012**

Creating an Accessible FAQ Page with Images

FAQ pages are among the most visited pages of many Web sites. Your FAQ may contain text, tables, links, images and titles, and these all should be accessible. Let's see how you can accomplish this. First, the HTML content should contain only semantic HTML tags, and any decoration elements should go in the stylesheet. So, instead of:

```
<i>italics</i>
```

you use:

```
<em>emphasized</em>
<cite>citation</cite>
```

and instead of:

```
<b>bold</b>
```

you use:

```
<strong>strong</strong>
```

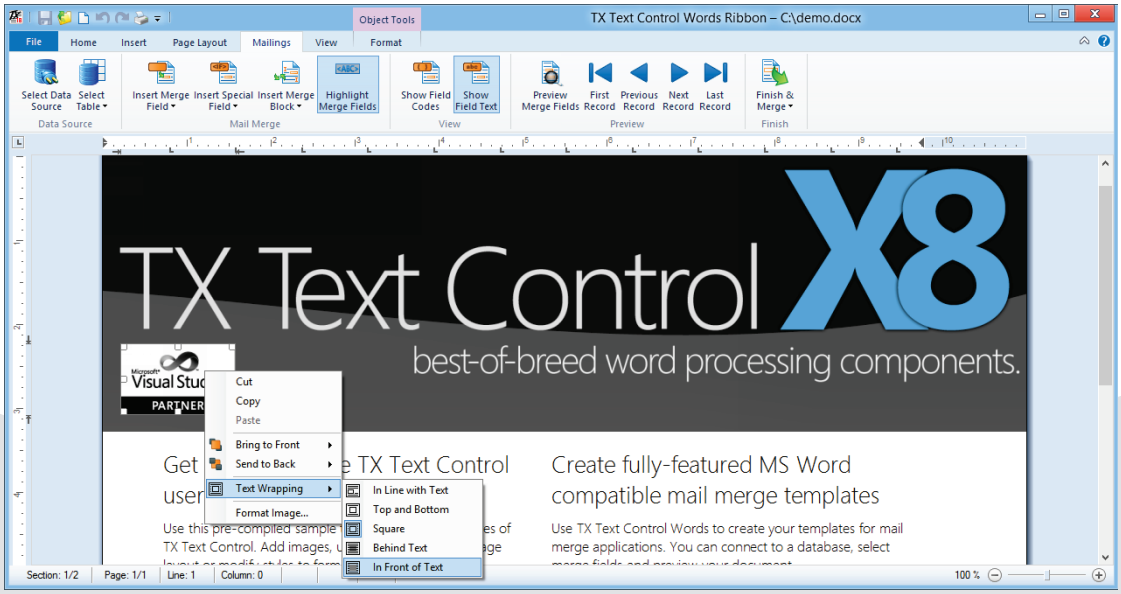
Note that these elements add meaning to the content and are interpreted differently by screen readers. For example, some screen readers will change the tone for the element but not for <b> elements.

WINDOWS FORMS | WPF | ASP.NET | ACTIVEX

WORD PROCESSING COMPONENTS

Ⓘ Rich Text Editing

Integrate professional rich text editing into your .NET based applications.



PDF Reflow - Open PDF Documents

Load, view, modify and convert Adobe PDF documents and reuse formatted text. Search and analyze documents such as invoices or delivery notes.

Free License

Download our 100% free Express version.

Spell Checking

Add the fastest spell checking engine to your Windows Forms, WPF or ASP.NET applications.

Reporting Redefined

Build MS Word compatible mail merge applications with Master-Detail views.

TX TEXT CONTROL® word processing components

Microsoft® Visual Studio® PARTNER

Word Processing Components for Windows Forms, WPF & ASP.NET

WWW.TEXTCONTROL.COM

US +1 877 - 462 - 4772 (toll-free)
EU +49 421 - 4270671 - 0

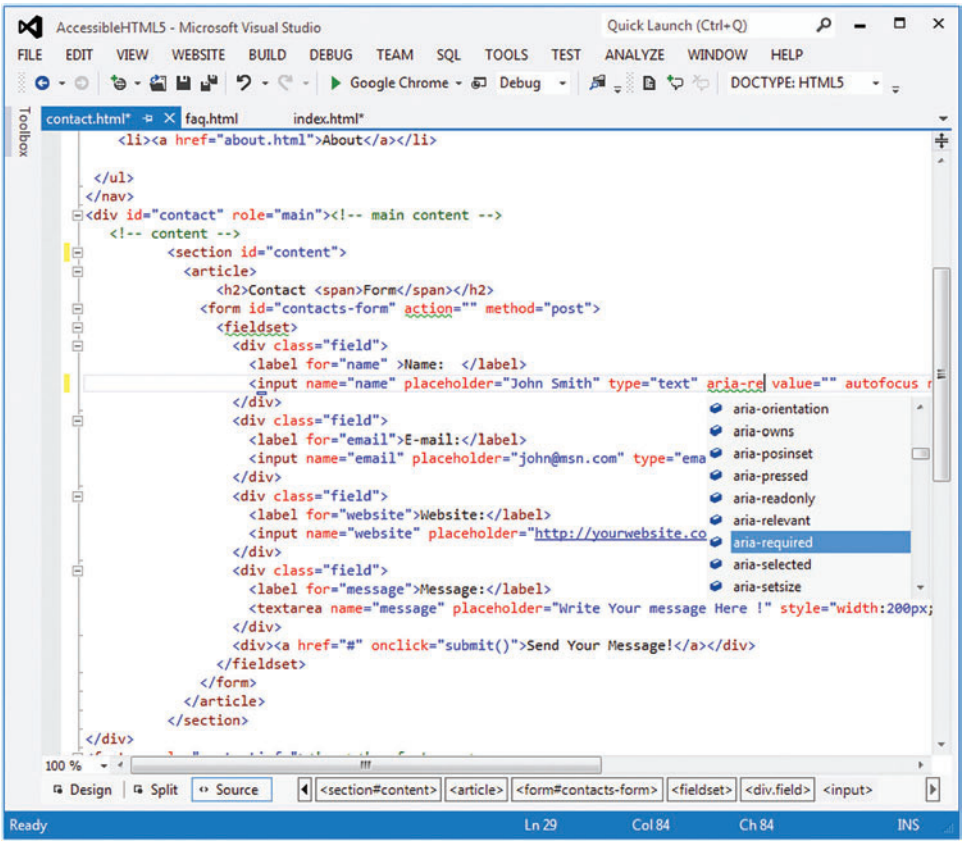


Figure 14 **ARIA Properties Are Supported in IntelliSense**

It's also important to properly use title-related heading elements such as <h1>, <h2> and so on. Ideally, you should use one <h1> heading in a page and multiple subheadings as required. Make sure you have closing tags for each HTML element. Also be sure to close all the Ordered and Unordered lists correctly. Standard practice also suggests you lowercase all tags and nest them correctly. **Figure 11** gives an example.

> Progressive enhancement is an approach to Web design that promotes accessibility using semantic HTML, stylesheets and scripting.

To create localized content, you set the language of the page using the lang attribute in the global <html> element:

```
<html lang="en">
```

For content in a different language, use lang within <p> or elements, like this:

```
<p><span lang="la">Carpe diem </span>(seize the day)</p>
```

And use the abbr tag for abbreviations:

```
<p>The <abbr title="World Wide Web Consortium">W3C</abbr> was founded in 1994.</p>
```

The font size of your content should always be relative. Never use absolute or fixed sizes, as this restricts the browser's font-scaling functionality. Use one of the following to increase or decrease the font size from the browser's default:

- percentage (%)
- em (relative to the size of the capital M)
- ex (relative to the size of the capital X)
- Keywords (small, medium, larger, smaller, larger and so forth)

Here's an example:

```
font-size:100%;
line-height:1.125em;
```

Color should be used as a visual aid to the content and should not be used alone for presenting information. A high color contrast between foreground and background is important to make the page accessible. The W3C recommends a contrast ratio of 4.5 to 1 for normal text and 3 to 1 for larger text.

For form validation, individual input elements may show the background as colored to indicate an error, which might not be visible to a color-blind user. Make sure you have multiple cues for the same information, such as a label to indicate an error.

When you're using a color in a stylesheet, set the background-color element to use a complementary color. Some people can read more easily with a black background, so allow change of page color to a darker theme. Here's an example:

```
body {
..font-family:Arial, Helvetica, sans-serif;
  font-size:100%;
  line-height:1.125em;
  background-color:#212222;
  color:#242424;
}
```

Standard tables generally have a header row and possibly a footer row, but distinguishing these isn't possible with simple table tags. HTML5, however, brings a number of helpful new tags:

- <caption> is the title of the table
- <details> shows additional details a user can view or hide on demand
- <summary> is announced before the real table data is read by a screen reader
- <thead> indicates the table header row
- <tfoot> indicates the table footer row

Figure 12 shows code for a sample HTML table that's accessible to AT devices.

When creating links, avoid using generic "click here" and "see more" links. Use the title attribute and meaningful anchor text. Here's a correct way to add links:

```
<p>Designandmethod.com has an article on accessibility. See the <a
title="click for more information at the Design & Method Web site"
href="http://designandmethod.com">Big picture at Design and Method</a></p>
```

Be careful about using ASCII symbols. When you have multiple pages, avoid using the greater-than and less-than symbols (> and <) to go forward and back to the next set of items. Instead, use clear text, such as "Next 10 items" and "Previous 10 items." Note that it may seem logical to use the greater-than symbol in breadcrumb navigation but, unfortunately, the screen reader will read "Next >>" as "Next, greater than, greater than," which is not useful. Use a CSS background image if your design requires a ">."

> Semantic HTML means the HTML tags in a page should describe the content in a way that has to do with its meaning rather than its presentation.

Finally, links should be underlined. This helps a color-blind user to determine that the text is a link. You can do this in the stylesheet using decoration:

```
{
  text-decoration: underline;
  display:block;
  border-bottom:1px solid #000;
}
```

To make images accessible, start with a meaningful alt attribute and use a blank alt (alt="") for decorative images. Be sure to include the title attribute—it shows up as a tool tip and is checked by screen readers if the alt attribute is not available. If neither the alt nor title attributes are found, the AT device will announce the name of the image, so be sure to give the image a meaningful name.

Use role=presentation for an image or for any element that's not relevant to AT devices. If you're including image maps, use alt for each area. You can also use the figure and figcaption tags for images. Keep animated images to a minimum; they can cause seizures for people with epilepsy.

The following example shows how to make an image accessible:

```
<figure><img src="images/
maingraphics.png" alt="Example
screen shot" title="main
graphics showing screenshot
example"><figcaption>Image with
caption.</figcaption></figure>
```

Here's the code for an image map:

```
<area shape=rect coords=0,0,10,10 href="example.htm" alt="example">
```

The last point on creating accessible content is to include relevant keywords at the beginning of titles. Having concise titles with relevant keywords at the beginning helps visually impaired users skim through them faster.

In the following example, having redundant "How to make" in each title seems more readable, but a screen reader would repeat those first three words for every title, making it difficult for users to skim the content quickly. Using precise and relevant keywords at the beginning of the title makes it more accessible:

```
<a href="#q1"><!--How to make content Accessible-->Accessible Content</a>
<a href="#q2"><!--How to make links Accessible-->Accessible Links</a>
<a href="#q3"><!--How to make images Accessible-->Accessible Image</a>
<a href="#q4"><!--How to make titles Accessible-->Accessible Titles</a>
```

Now let's take a quick look at adding an About page to the accessible Web site—one that contains audio and video. Let's see what I need to make audio and video elements accessible.

An Accessible About Page with Audio and Video

Suppose I want my About page to contain a video that explains the founding of my Web site. The <audio> and <video> tags in HTML5 make it easy to embed multimedia content in a Web page, but doing so creates challenges with respect to accessing content for those who are non-native speakers, deaf or hard of hearing, blind, or for anyone who may have broken speakers or be in a loud environment. There are specific guidelines to follow for each of these challenges. Here are some ways to make audio and video content accessible:

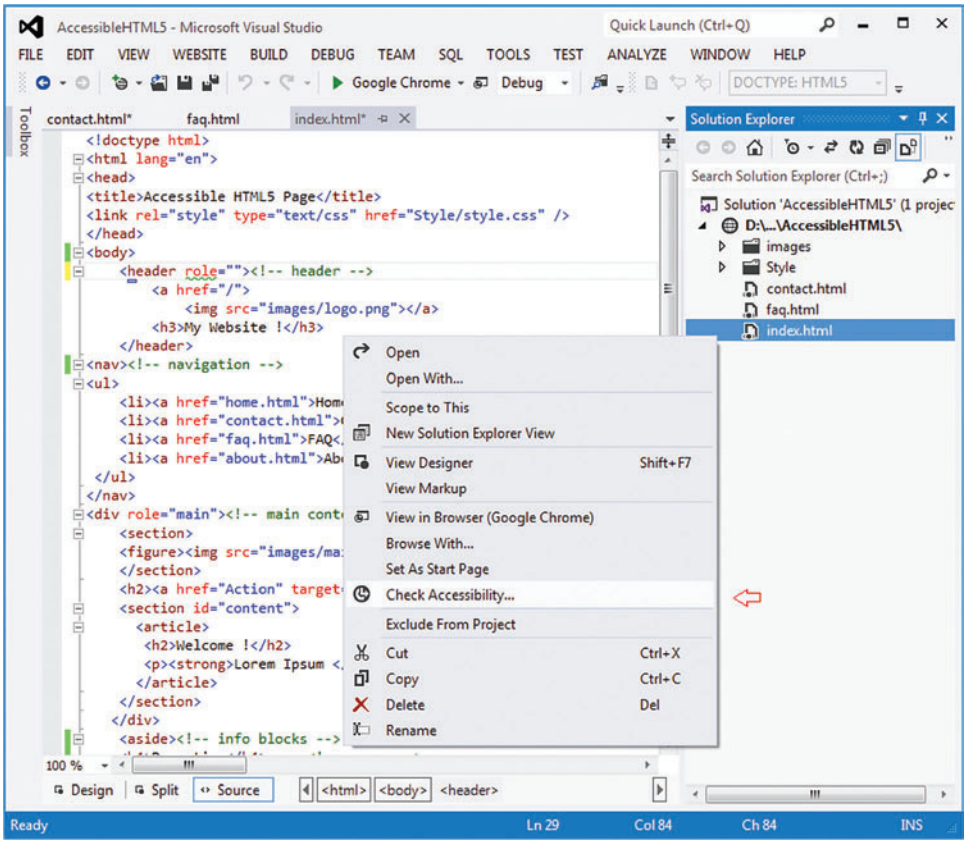


Figure 15 **You Can Check Accessibility of a Web Page in Visual Studio 2012**

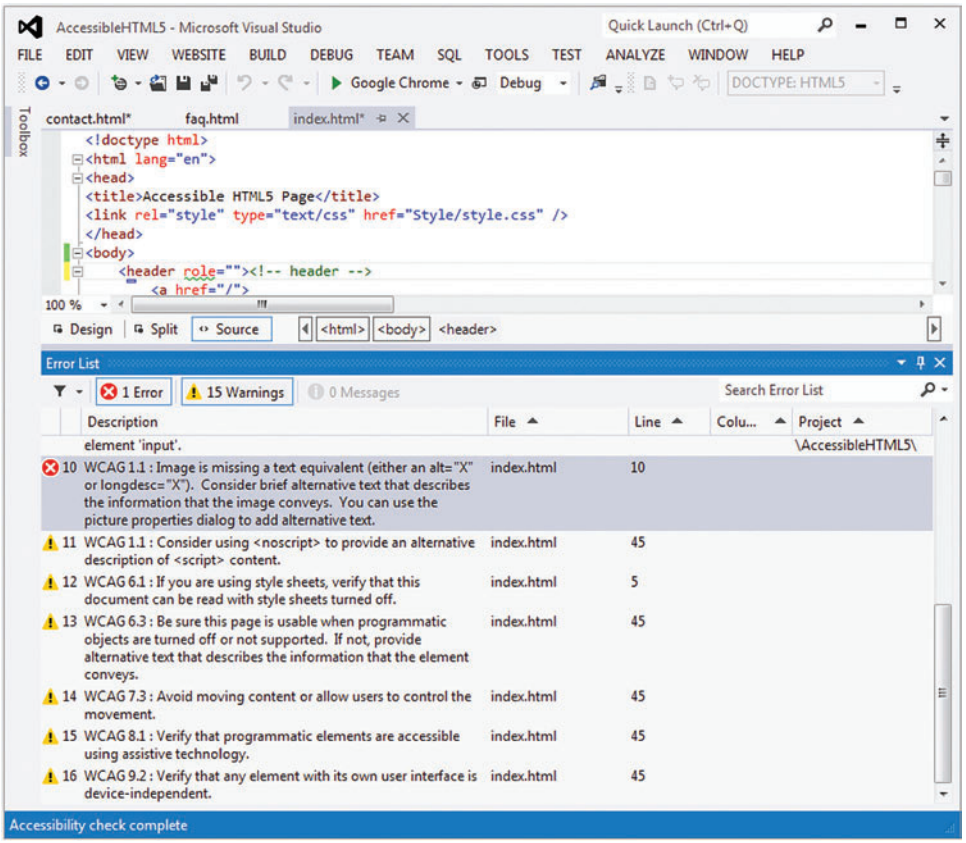


Figure 16 **Web Content Accessibility Report in Visual Studio 2012**

- Include text transcripts for the audio or video in HTML format.
- Include alternative content for browsers that don't support media tags.
- Controls should, at minimum, have an On/Off button.
- Media should not auto-start but should be user-initiated.
- Provide a link to download the media file.
- Provide captioning (closed captions or subtitles) using a video/audio track.

Subtitles are typically a time-aligned transcription of the spoken words in a video, which can help users to understand the content. For deaf users, captions are a better solution because they include transcriptions of noises, sound effects, music and the like, along with spoken words.

Right now we have caption and subtitle support using the track element along with the following formats:

- WebVTT for captioning video content
- SMPTE-TT, a timed-text format for subtitling
- SRT, for subtitle files

Note that accessibility support for video is still a work in progress, but here's an example:

```
<video controls>
<source src="video-file.mp4" type="video/mp4"/>
<track src="en.vtt" kind="subtitles" srclang="en"
label="English p subtitles" default/>
<track src="en.ttml" kind="captions" srclang="en"
label="English p captions" default/>
</video>
```

Although there isn't granular control over media, HTML5 has a controls attribute that displays controls for the media element. These controls are accessible by keyboard:

- The space bar toggles between play and pause.
- The left and right arrows wind the video forward and back by 5 seconds.
- CTRL+left arrow or right arrow winds the video forward or back by 60 seconds.
- HOME+left arrow or right arrow jumps to the beginning or end of the video.
- If the volume button has focus, the up and down arrows increase and decrease the volume.

Accessibility Support in Visual Studio 2012

You'll be happy to know that Visual Studio 2012 makes accessibility easier. There's now IntelliSense for ARIA roles, attributes and properties in HTML elements, as shown in **Figure 13** and **Figure 14**.

After you create an accessible Web page, you'll want to check to make sure it actually meets accessibility requirements. With Visual Studio 2012, you can do so easily by right-clicking on a page and selecting Check Accessibility, as shown in **Figure 15**.

You then choose the Web Content Accessibility Guidelines, or WCAG, level you want to check against: Priority 1 or Priority 2 (see bit.ly/SONb66). You can also check against Access Board Section 508, which refers to standards defined by the United States government in Section 508 of the Rehabilitation Act (section508.gov). Once you've selected the guidelines, Visual Studio 2012 checks all the HTML elements and displays a detailed report on any errors or warnings in the page, as shown in **Figure 16**.

> All modern OSes have their own accessibility APIs.

You've now learned how to create an accessible Web site using HTML5, and you've seen how some Visual Studio 2012 accessibility features can help. These should be useful additions to your toolbox as you explore the accessible Web. ■

RAJESH LAL *works at Nokia and is passionate about HTML5 and Web technologies. He has written multiple books on Windows gadgets, Web widgets, the mobile Web and Silverlight technologies. To discuss accessibility in software design and method, visit dsgnmthd.com/accessibility. For information about the author, check out iRajLal.com.*

THANKS to the following technical experts for reviewing this article:
Art Barstow, Lakshmi C. Chava and Dennis Lembrée

sponsored by Microsoft

ALM Summit 3

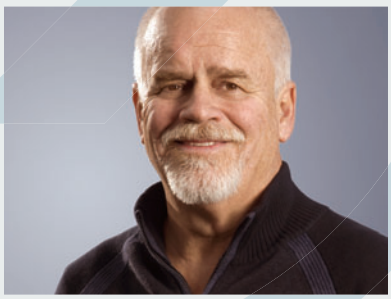
expertise worth sharing

January 29-31, 2013 - Microsoft Conference Center, Redmond

Register at www.alm-summit.com

follow us on

keynote speakers



Dean Leffingwell
Creator, Scaled Agile Framework
"Being Agile. Scaling Up. Staying Lean"

Scott Porad
CTO, Cheezburger Network
"Lean Startups, Cheezburgers, and You"

Brian Harry
Technical Fellow, Microsoft
"Building an Engineering Organization for Continuous Delivery"

Jason Zander
Corporate Vice President, Microsoft
"Continuous Services and Connected Devices"

Sam Guckenheimer
Product Owner, Microsoft
"Reimagining the Application Lifecycle"

James Whittaker
Testing Guru, Microsoft
"Re-inventing the Internet"

conference tracks

ALM Leadership • Agile Development • DevOps • Testing

sponsors

Visual Studio
diamond

Scrum.org
Improving the Profession of Software Development
platinum

NORTHWEST CADENCE
where technology meets teamwork

INCYCLE SOFTWARE

The C# Memory Model in Theory and Practice

Igor Ostrovsky

This is the first of a two-part series that will tell the long story of the C# memory model. The first part explains the guarantees the C# memory model makes and shows the code patterns that motivate the guarantees; the second part will detail how the guarantees are achieved on different hardware architectures in the Microsoft .NET Framework 4.5.

One source of complexity in multithreaded programming is that the compiler and the hardware can subtly transform a program's memory operations in ways that don't affect the single-threaded behavior, but might affect the multithreaded behavior. Consider the following method:

```
void Init() {
  _data = 42;
  _initialized = true;
}
```

If _data and _initialized are ordinary (that is, non-volatile) fields, the compiler and the processor are allowed to reorder the operations so that Init executes as if it were written like this:

```
void Init() {
  _initialized = true;
  _data = 42;
}
```

This article discusses:

- The C# language specification
- Memory operation reordering
- Thread communication patterns
- Interlocked operations and memory barriers

Technologies discussed:

C#

There are various optimizations in both compilers and processors that can result in this kind of reordering, as I'll discuss in Part 2.

In a single-threaded program, the reordering of statements in Init makes no difference in the meaning of the program. As long as both _initialized and _data are updated before the method returns, the order of the assignments doesn't matter. In a single-threaded program, there's no second thread that could observe the state in between the updates.

> Even if the compiler and the processor are allowed to reorder memory operations, it doesn't mean they always do so in practice.

In a multithreaded program, however, the order of the assignments may matter because another thread might read the fields while Init is in the middle of execution. Consequently, in the reordered version of Init, another thread may observe _initialized=true and _data=0.

The C# memory model is a set of rules that describes what kinds of memory-operation reordering are and are not allowed. All programs should be written against the guarantees defined in the specification.

Figure 1 **Code at Risk of Memory Operation Reordering**

```
public class DataInit {
  private int _data = 0;
  private bool _initialized = false;

  void Init() {
    _data = 42;            // Write 1
    _initialized = true;   // Write 2
  }

  void Print() {
    if (_initialized)           // Read 1
      Console.WriteLine(_data); // Read 2
    else
      Console.WriteLine("Not initialized");
  }
}
```

However, even if the compiler and the processor are allowed to reorder memory operations, it doesn't mean they always do so in practice. Many programs that contain a "bug" according to the abstract C# memory model will still execute correctly on particular hardware running a particular version of the .NET Framework. Notably, the x86 and x64 processors reorder operations only in certain narrow scenarios, and similarly the CLR just-in-time (JIT) compiler doesn't perform many of the transformations it's allowed to.

> The C# memory model permits reordering of memory operations in a method, as long as the behavior of single-threaded execution doesn't change.

Although the abstract C# memory model is what you should have in mind when writing new code, it can be helpful to understand the actual implementation of the memory model on different architectures, in particular when trying to understand the behavior of existing code.

C# Memory Model According to ECMA-334

The authoritative definition of the C# memory model is in the Standard ECMA-334 C# Language Specification (bit.ly/MXMCrN). Let's discuss the C# memory model as defined in the specification.

Memory Operation Reordering According to ECMA-334, when a thread reads a memory location in C# that was written to by a different thread, the reader might see a stale value. This problem is illustrated in **Figure 1**.

Suppose Init and Print are called in parallel (that is, on different threads) on a new instance of DataInit. If you examine the code of Init and Print, it may seem that Print can only output "42" or "Not initialized." However, Print can also output "0."

The C# memory model permits reordering of memory operations in a method, as long as the behavior of single-threaded execution doesn't change. For example, the compiler and the processor are free to reorder the Init method operations as follows:

```
void Init() {
  _initialized = true;   // Write 2
  _data = 42;            // Write 1
}
```

This reordering wouldn't change the behavior of the Init method in a single-threaded program. In a multithreaded program, however, another thread might read _initialized and _data fields after Init has modified one field but not the other, and then the reordering could change the behavior of the program. As a result, the Print method could end up outputting a "0."

The reordering of Init isn't the only possible source of trouble in this code sample. Even if the Init writes don't end up reordered, the reads in the Print method could be transformed:

```
void Print() {
  int d = _data;     // Read 2
  if (_initialized)  // Read 1
    Console.WriteLine(d);
  else
    Console.WriteLine("Not initialized");
}
```

Just as with the reordering of writes, this transformation has no effect in a single-threaded program, but might change the behavior of a multithreaded program. And, just like the reordering of writes, the reordering of reads can also result in a 0 printed to the output.

In Part 2 of this article, you'll see how and why these transformations take place in practice when I look at different hardware architectures in detail.

Volatile Fields The C# programming language provides volatile fields that constrain how memory operations can be reordered. The ECMA specification states that volatile fields provide acquire-release semantics (bit.ly/NArSIt).

A read of a volatile field has acquire semantics, which means it can't be reordered with subsequent operations. The volatile read forms a one-way fence: preceding operations can pass it, but subsequent operations can't. Consider this example:

```
class AcquireSemanticsExample {
  int _a;
  volatile int _b;
  int _c;

  void Foo() {
    int a = _a; // Read 1
    int b = _b; // Read 2 (volatile)
    int c = _c; // Read 3
    ...
  }
}
```

Figure 2 **Valid Reordering of Reads in AcquireSemanticsExample**

```
int a = _a; // Read 1
int b = _b; // Read 2 (volatile)
int c = _c; // Read 3
```

```
int b = _b; // Read 2 (volatile)
int a = _a; // Read 1
int c = _c; // Read 3
```

```
int b = _b; // Read 2 (volatile)
int c = _c; // Read 3
int a = _a; // Read 1
```

Figure 3 **Valid Reordering of Writes in ReleaseSemanticsExample**

```
_a = 1; // Write 1
_b = 1; // Write 2 (volatile)
_c = 1; // Write 3
```

```
_a = 1; // Write 1
_c = 1; // Write 3
_b = 1; // Write 2 (volatile)
```

```
_c = 1; // Write 3
_a = 1; // Write 1
_b = 1; // Write 2 (volatile)
```

Figure 4 **Thread Communication with Locking**

```
public class Test {
  private int _a = 0;
  private int _b = 0;
  private object _lock = new object();

  void Set() {
    lock (_lock) {
      _a = 1;
      _b = 1;
    }
  }

  void Print() {
    lock (_lock) {
      int b = _b;
      int a = _a;
      Console.WriteLine("{0} {1}", a, b);
    }
  }
}
```

Read 1 and Read 3 are non-volatile, while Read 2 is volatile. Read 2 can't be reordered with Read 3, but it can be reordered with Read 1. **Figure 2** shows the valid reorderings of the Foo body.

A write of a volatile field, on the other hand, has release semantics, and so it can't be reordered with prior operations. A volatile write forms a one-way fence, as this example demonstrates:

```
class ReleaseSemanticsExample
{
  int _a;
  volatile int _b;
  int _c;

  void Foo()
  {
    _a = 1; // Write 1
    _b = 1; // Write 2 (volatile)
    _c = 1; // Write 3
    ...
  }
}
```

Write 1 and Write 3 are non-volatile, while Write 2 is volatile. Write 2 can't be reordered with Write 1, but it can be reordered with Write 3. **Figure 3** shows the valid reorderings of the Foo body.

I'll come back to the acquire-release semantics in the "Publication via Volatile Field" section later in this article.

Atomicity Another issue to be aware of is that in C#, values aren't necessarily written atomically into memory. Consider this example:

```
class AtomicityExample {
  Guid _value;

  void SetValue(Guid value) { _value = value; }
  Guid GetValue() { return _value; }
}
```

If one thread repeatedly calls SetValue and another thread calls GetValue, the getter thread might observe a value that was never written by the setter thread. For example, if the setter thread alternately calls SetValue with Guid values (0,0,0,0) and (5,5,5,5), GetValue could observe (0,0,0,5) or (0,0,5,5) or (5,5,0,0), even though none of those values was ever assigned using SetValue.

The reason behind the "tearing" is that the assignment "_value = value" doesn't execute atomically at the hardware level. Similarly, the read of _value also doesn't execute atomically.

The C# ECMA specification guarantees that the following types will be written atomically: reference types, bool, char, byte, sbyte, short, ushort, uint, int and float. Values of other types—including user-defined value types—could be written into memory in multiple atomic writes. As a result, a reading thread could observe a *torn* value consisting of pieces of different values.

One caveat is that even the types that are normally read and written atomically (such as int) could be read or written non-atomically if the value is not *correctly aligned* in memory. Normally, C# will ensure that values are correctly aligned, but the user is able to override the alignment using the StructLayoutAttribute class (bit.ly/TqaOMZ).

Non-Reordering Optimizations Some compiler optimizations may introduce or eliminate certain memory operations. For example, the compiler might replace repeated reads of a field with a single read. Similarly, if code reads a field and stores the value in a local variable and then repeatedly reads the variable, the compiler could choose to repeatedly read the field instead.

Because the ECMA C# spec doesn't rule out the non-reordering optimizations, they're presumably allowed. In fact, as I'll discuss in Part 2, the JIT compiler does perform these types of optimizations.

One caveat is that even the types that are normally read and written atomically (such as int) could be read or written non-atomically if the value is not correctly aligned in memory.

Thread Communication Patterns

The purpose of a memory model is to enable thread communication. When one thread writes values to memory and another thread reads from memory, the memory model dictates what values the reading thread might see.

Locking Locking is typically the easiest way to share data among threads. If you use locks correctly, you basically don't have to worry about any of the memory model messiness.

Whenever a thread acquires a lock, the CLR ensures that the thread will see all updates made by the thread that held the lock earlier. Let's add locking to the example from the beginning of this article, as shown in **Figure 4**.

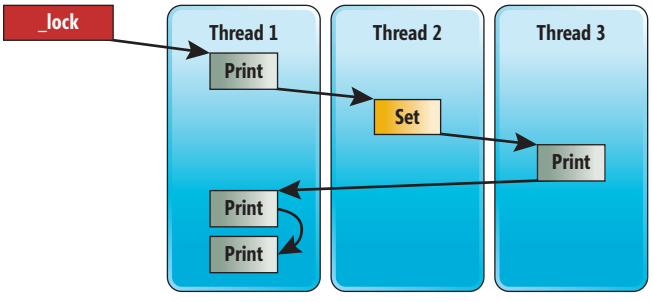


Figure 5 **Sequential Execution with Locking**

Visual Studio LIVE!
EXPERT SOLUTIONS FOR .NET DEVELOPERS

2013 LIVE!
CELEBRATING TWENTY YEARS

YOUR BACKSTAGE PASS TO LIVE! THE MICROSOFT PLATFORM

VISUAL STUDIO LIVE! IS COMING TO A CITY NEAR YOU

PICK YOUR TOUR DATES

Get an all-access look at the Microsoft Platform and practical, unbiased Developer training at **Visual Studio Live!**. Pick your tour dates and join .NET rockstars for a week of educational sessions, workshops and networking events.

HTML5 | Windows 8 | Visual Studio/.NET | Mobile | WPF/ Silverlight

There are FOUR tour dates / locations to choose from – pick your favorite and prepare to solve your development challenges in 2013!

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

SUPPORTED BY
Microsoft
Visual Studio
msdn magazine
Visual Studio MAGAZINE

PRODUCED BY
1105 MEDIA

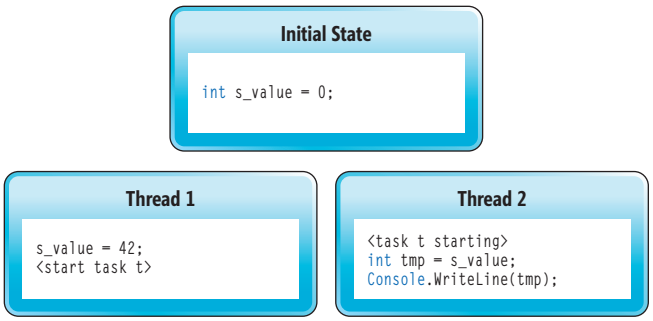


Figure 6 **Two Threads Communicating via a Non-Volatile Field**

Adding a lock that Print and Set acquire provides a simple solution. Now, Set and Print execute atomically with respect to each other. The lock statement guarantees that the bodies of Print and Set will appear to execute in *some* sequential order, even if they're called from multiple threads.

The diagram in **Figure 5** shows one possible sequential order that could happen if Thread 1 calls Print three times, Thread 2 calls Set once and Thread 3 calls Print once.

> Locking is a very general and powerful mechanism for sharing state among threads.

When a locked block of code executes, it's guaranteed to see all writes from blocks that precede the block in the sequential order of the lock. Also, it's guaranteed *not to see* any of the writes from blocks that follow it in the sequential order of the lock.

In short, locks hide all of the unpredictability and complexity weirdness of the memory model: You don't have to worry about the reordering of memory operations if you use locks correctly. However, note that the use of locking has to be correct. If only Print *or* Set uses the lock—or Print and Set acquire two different locks—memory operations can become reordered and the complexity of the memory model comes back.

Figure 7 **Using the Volatile Keyword**

```
public class DataInit {
  private int _data = 0;
  private volatile bool _initialized = false;

  void Init() {
    _data = 42;           // Write 1
    _initialized = true;  // Write 2
  }

  void Print() {
    if (_initialized) {         // Read 1
      Console.WriteLine(_data); // Read 2
    }
    else {
      Console.WriteLine("Not initialized");
    }
  }
}
```

Publication via Threading API Locking is a very general and powerful mechanism for sharing state among threads. Publication via threading API is another frequently used pattern of concurrent programming.

The easiest way to illustrate publication via threading API is by way of an example:

```
class Test2 {
  static int s_value;

  static void Run() {
    s_value = 42;
    Task t = Task.Factory.StartNew(() => {
      Console.WriteLine(s_value);
    });
    t.Wait();
  }
}
```

When you examine the preceding code sample, you'd probably expect "42" to be printed to the screen. And, in fact, your intuition would be correct. This code sample is guaranteed to print "42."

It might be surprising that this case even needs to be mentioned, but in fact there are possible implementations of StartNew that would allow "0" to be printed instead of "42," at least in theory. After all, there are two threads communicating via a non-volatile field, so memory operations can be reordered. The pattern is displayed in the diagram in **Figure 6**.

The StartNew implementation must ensure that the write to s_value on Thread 1 will not move after <start task t>, and the read from s_value on Thread 2 will not move before <task t starting>. And, in fact, the StartNew API really does guarantee this.

All other threading APIs in the .NET Framework, such as Thread.Start and ThreadPool.QueueUserWorkItem, also make a similar guarantee. In fact, nearly every threading API must have some barrier semantics in order to function correctly. These are almost never documented, but can usually be deduced simply by thinking about what the guarantees would have to be in order for the API to be useful.

Publication via Type Initialization Another way to safely publish a value to multiple threads is to write the value to a static field in a static initializer or a static constructor. Consider this example:

```
class Test3
{
  static int s_value = 42;
  static object s_obj = new object();

  static void PrintValue()
  {
    Console.WriteLine(s_value);
    Console.WriteLine(s_obj == null);
  }
}
```

If Test3.PrintValue is called from multiple threads concurrently, is it guaranteed that each PrintValue call will print "42" and "false"? Or, could one of the calls also print "0" or "true"? Just as in the previous case, you do get the behavior you'd expect: Each thread is guaranteed to print "42" and "false."

The patterns discussed so far all behave as you'd expect. Now I'll get to cases whose behavior may be surprising.

Publication via Volatile Field Many concurrent programs can be built using the three simple patterns discussed so far, used together with concurrency primitives in the .NET System.Threading and System.Collections.Concurrent namespaces.

MSDN Magazine Online

It's like *MSDN Magazine*—only better. In addition to all the great articles from the print edition, you get:

- Code Downloads
- The *MSDN Magazine* Blog
- Digital Magazine Downloads
- Searchable Content

All of this and more at msdn.microsoft.com/magazine

msdn magazine

Figure 8 **Lazy Initialization**

```
class BoxedInt
{
  public int Value { get; set; }
}

class LazyInit
{
  volatile BoxedInt _box;

  public int LazyGet()
  {
    var b = _box;  // Read 1
    if (b == null)
    {
      lock(this)
      {
        b = new BoxedInt();
        b.Value = 42; // Write 1
        _box = b;     // Write 2
      }
    }
    return b.Value; // Read 2
  }
}
```

The pattern I'm about to discuss is so important that the semantics of the *volatile* keyword were designed around it. In fact, the best way to remember the volatile keyword semantics is to remember this pattern, instead of trying to memorize the abstract rules explained earlier in this article.

Let's start with the example code in **Figure 7**. The DataInit class in **Figure 7** has two methods, Init and Print; both may be called from multiple threads. If no memory operations are reordered, Print can only print "Not initialized" or "42," but there are two possible cases when Print could print a "0":

- Write 1 and Write 2 were reordered.
- Read 1 and Read 2 were reordered.

If _initialized were not marked as volatile, both reorderings would be permitted. However, when _initialized is marked as volatile, neither reordering is allowed! In the case of writes, you have an ordinary write followed by a volatile write, and a volatile write can't be reordered with a prior memory operation. In the case of the reads, you have a volatile read followed by an ordinary read, and a volatile read can't be reordered with a subsequent memory operation.

So, Print will never print "0," even if called concurrently with Init on a new instance of DataInit.

Figure 9 **Broken Polling Loop**

```
class PollingLoopExample
{
  private bool _loop = true;

  public static void Main()
  {
    PollingLoopExample test1 = new PollingLoopExample();

    // Set _loop to false on another thread
    new Thread(() => { test1._loop = false;}).Start();

    // Poll the _loop field until it is set to false
    while (test1._loop) ;

    // The previous loop may never terminate
  }
}
```

Note that if the _data field is volatile but _initialized is not, both reorderings would be permitted. As a result, remembering this example is a good way to remember the volatile semantics.

Lazy Initialization One common variant of publication via volatile field is lazy initialization. The example in **Figure 8** illustrates lazy initialization.

In this example, LazyGet is always guaranteed to return "42." However, if the _box field were not volatile, LazyGet would be allowed to return "0" for two reasons: the reads could get reordered, or the writes could get reordered.

To further emphasize the point, consider this class:

```
class BoxedInt2
{
  public readonly int _value = 42;

  void PrintValue()
  {
    Console.WriteLine(_value);
  }
}
```

Now, it's possible—at least in theory—that PrintValue will print "0" due to a memory-model issue. Here's a usage example of BoxedInt that allows it:

```
class Tester
{
  BoxedInt2 _box = null;

  public void Set() {
    _box = new BoxedInt2();
  }

  public void Print() {
    var b = _box;
    if (b != null) b.PrintValue();
  }
}
```

Because the BoxedInt instance was incorrectly published (through a non-volatile field, _box), the thread that calls Print may observe a partially constructed object! Again, making the _box field volatile would fix the issue.

Interlocked Operations and Memory Barriers Interlocked operations are atomic operations that can be used at times to reduce locking in a multithreaded program. Consider this simple thread-safe counter class:

```
class Counter
{
  private int _value = 0;
  private object _lock = new object();

  public int Increment()
  {
    lock (_lock)
    {
      _value++;
      return _value;
    }
  }
}
```

Using Interlocked.Increment, you can rewrite the program like this:

```
class Counter
{
  private int _value = 0;

  public int Increment()
  {
    return Interlocked.Increment(ref _value);
  }
}
```

As rewritten with Interlocked.Increment, the method should execute faster, at least on some architectures. In addition to the increment operations, the Interlocked class (bit.ly/RksCMF) exposes methods for various atomic operations: adding a value, conditionally replacing a value, replacing a value and returning the original value, and so forth.

> All Interlocked methods have one very interesting property: They can't be reordered with other memory operations.

All Interlocked methods have one very interesting property: They can't be reordered with other memory operations. So no memory operation, whether before or after an Interlocked operation, can pass an Interlocked operation.

An operation that's closely related to Interlocked methods is Thread.MemoryBarrier, which can be thought of as a dummy Interlocked operation. Just like an Interlocked method, Thread.MemoryBarrier can't be reordered with any prior or subsequent memory operations. Unlike an Interlocked method, though, Thread.MemoryBarrier has no side effect; it simply constrains memory reorderings.

Polling Loop Polling loop is a pattern that's generally not recommended but—somewhat unfortunately—frequently used in practice. **Figure 9** shows a broken polling loop.

In this example, the main thread loops, polling a particular non-volatile field. A helper thread sets the field in the meantime, but the main thread may never see the updated value.

Now, what if the _loop field was marked volatile? Would that fix the program? The general expert consensus seems to be that the compiler isn't allowed to hoist a volatile field read out of a loop, but it's debatable whether the ECMA C# specification makes this guarantee.

Figure 10 **Concurrency Primitives in the .NET Framework 4**

Type	Description
Lazy<>	Lazily initialized values
LazyInitializer	
BlockingCollection<>	Thread-safe collections
ConcurrentBag<>	
ConcurrentDictionary<,>	
ConcurrentQueue<>	
ConcurrentStack<>	
AutoResetEvent	Primitives to coordinate execution of different threads
Barrier	
CountdownEvent	
ManualResetEventSlim	
Monitor	
SemaphoreSlim	
ThreadLocal<>	Container that holds a separate value for every thread

Best Practices

- All code you write should rely only on the guarantees made by the ECMA C# specification, and not on any of the implementation details explained in this article.
- Avoid unnecessary use of volatile fields. Most of the time, locks or concurrent collections (System.Collections.Concurrent.*) are more appropriate for exchanging data between threads. In some cases, volatile fields can be used to optimize concurrent code, but you should use performance measurements to validate that the benefit outweighs the extra complexity.
- Instead of implementing the lazy initialization pattern yourself using a volatile field, use the System.Lazy<T> and System.Threading.LazyInitializer types.
- Avoid polling loops. Often, you can use a BlockingCollection<T>, Monitor.Wait/Pulse, events or asynchronous programming instead of a polling loop.
- Whenever possible, use the standard .NET concurrency primitives instead of implementing equivalent functionality yourself.

On one hand, the specification states only that volatile fields obey the acquire-release semantics, which doesn't seem sufficient to prevent hoisting of a volatile field. On the other hand, the example code in the specification does in fact poll a volatile field, implying that the volatile field read can't be hoisted out of the loop.

On x86 and x64 architectures, PollingLoopExample.Main will typically hang. The JIT compiler will read test1._loop field just once, save the value in a register, and then loop until the register value changes, which will obviously never happen.

If the loop body contains some statements, however, the JIT compiler will probably need the register for some other purpose, so each iteration may end up rereading test1._loop. As a result, you may end up seeing loops in existing programs that poll a non-volatile field and yet happen to work.

Concurrency Primitives Much concurrent code can benefit from high-level concurrency primitives that became available in the .NET Framework 4. **Figure 10** lists some of the .NET concurrency primitives.

By using these primitives, you can often avoid low-level code that depends on the memory model in intricate ways (via volatile and the like).

Coming Up

So far, I've described the C# memory model as defined in the ECMA C# specification, and discussed the most important patterns of thread communication that define the memory model.

The second part of this article will explain how the memory model is actually implemented on different architectures, which is helpful for understanding the behavior of programs in the real world. ■

Igor Ostrovsky *is a senior software development engineer at Microsoft. He has worked on Parallel LINQ, the Task Parallel Library, and other parallel libraries and primitives in the Microsoft .NET Framework. Ostrovsky blogs about programming topics at igoro.com.*

Thanks to the following technical expert for reviewing this article: *Joe Duffy, Eric Eilebrecht, Joe Hoag, Emad Omara, Grant Richins, Jaroslav Sevcik and Stephen Toub*

Matrix Decomposition

James McCaffrey

Matrix decomposition, a technique that breaks down a square numeric matrix into two different square matrices, is the basis for efficiently solving a system of equations, which in turn is the basis for inverting a matrix. And inverting a matrix is a part of many important algorithms. This article presents and explains C# code that performs matrix decomposition, matrix inversion, a system of equations solution and related operations.

Admittedly, matrix decomposition isn't a flashy topic, but a collection of matrix methods can be an important addition to your personal code library. The methods are explained so you can modify the source code to meet your own needs. Additionally, some of the techniques used in the matrix methods can be reused in other coding scenarios.

The best way for you to get a feel for the kind of information presented in this article is to take a look at the screenshot in **Figure 1**.

This article discusses:

- Implementing a matrix in C#
- Parallelizing matrix multiplication
- Approaches to matrix decomposition
- Using matrix decomposition to invert a matrix
- Computing the determinant of a matrix

Technologies discussed:

C#, Microsoft .NET Framework

Code download available at:

archive.msdn.microsoft.com/mag201212Matrix

The demo program begins by creating a 4 x 4 square matrix and displaying its values. Next, the matrix is decomposed into what's called an LUP matrix. L stands for lower and U stands for upper. The P part (P stands for permutation) is an array with values {3,1,2,0} and indicates that rows 0 and 3 were exchanged during the decomposition process. The decomposition also generated a toggle value of -1, indicating that an odd number of row exchanges occurred. The demo program displays the decomposition in two ways: first as a combined LU matrix and then as separate L and U matrices. Next, the program computes and displays the inverse of the original matrix, using the LUP matrix behind the scenes. The demo program computes the determinant of the original matrix, again using the decomposition. It then uses the inverse of the matrix to solve a system of linear equations and concludes by combining the L and U matrices back into the original matrix.

But why go to all the trouble of creating a custom matrix decomposition method and a library of related methods? Although there are many standalone matrix tools available, they can sometimes be difficult to integrate into an application or system. And in spite of the fundamental importance of matrix decomposition, there are few free, non-copyrighted .NET code implementations available; those that do exist are often not explained in enough detail for you to modify the source code to suit your coding scenarios.

This article assumes you have intermediate C# programming skills and at least a basic understanding of matrix operations and terminology. All of the key C# code is presented in this article. The code is also available from the MSDN code download site at code.msdn.microsoft.com/mag201207matrix.

Matrix Definition

There are several ways to implement a matrix in C#. The traditional approach, and the one used in this article, is to use an array of arrays, sometimes called a jagged array. For example, this code defines a matrix with three rows and two columns:

```
double[][] m = new double[3][];
m[0] = new double[2];
m[1] = new double[2];
m[2] = new double[2];
m[2][1] = 5.0; // set row 2, col 1
```

Unlike most programming languages, C# has a built-in multidimensional array type, which provides an alternative approach. For example:

```
double[,] m = new double[3,2];
m[2,1] = 5.0;
```

A third approach to implementing matrices in C# is to use a single array combined with array index manipulation, like this:

```
int rows = 3;
int cols = 2;
double[] m = new double[rows * cols];
int i = 2;
int j = 1;
m[i * cols + j] = 5.0;
```

Regardless of the storage scheme used, matrices can be implemented using either an OOP or a static-method approach. For example, an OOP approach could resemble:

```
public class MyMatrix
{
  int m; // number rows
  int n; // number columns
  data[][]; // the values
  ...
}
```

There's no single best choice for matrix design; the best design depends on the particular coding scenario you're operating in and on your personal coding preferences. This article uses a static-method approach because it's the easiest to understand and refactor.

When using an array-of-arrays design for matrices, because each row must be allocated separately, it's often convenient to define a helper method to perform memory allocation. For example:

```
static double[][] MatrixCreate(int rows, int cols)
{
  // creates a matrix initialized to all 0.0s
  // do error checking here?
  double[][] result = new double[rows][];
  for (int i = 0; i < rows; ++i)
    result[i] = new double[cols]; // auto init to 0.0
  return result;
}
```

The method can be called like so:

```
double[][] m = MatrixCreate(3,2);
m[2][1] = 5.0;
```

This method demonstrates one advantage of creating your own library of matrix methods: If you want to improve performance, you can omit error-checking the input parameters—at the expense of increasing the risk of the calling code causing an exception. (To keep this article short, most error checking has been removed.)

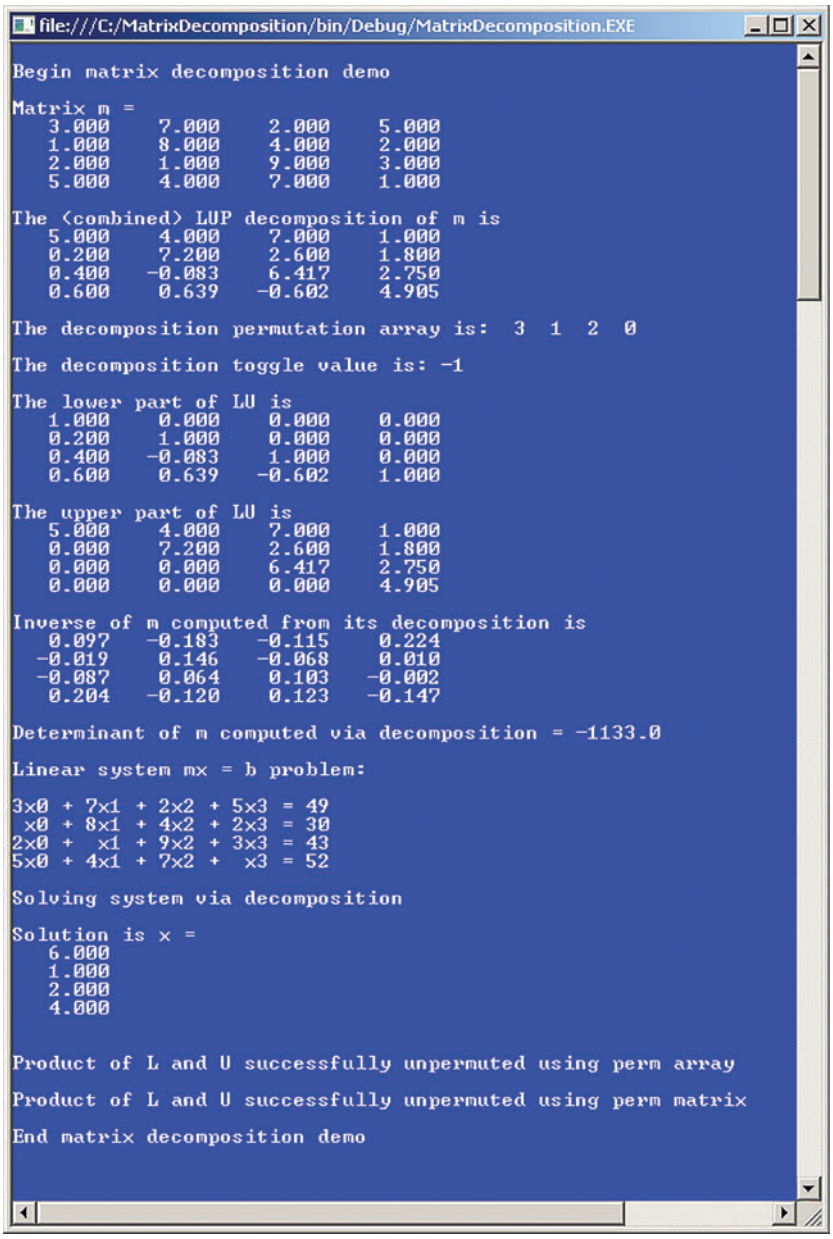


Figure 1 **Matrix Decomposition Demo**

Another advantage is that you can customize your library to optimize for your exact scenario. The main disadvantage of creating your own library is that it can take longer than using an existing library.

Another convenient method to add to your matrix library is one that can be used to display a matrix as a string. Here's one possibility:

```
static string MatrixAsString(double[][] matrix)
{
  string s = "";
  for (int i = 0; i < matrix.Length; ++i)
  {
    for (int j = 0; j < matrix[i].Length; ++j)
      s += matrix[i][j].ToString("F3").PadLeft(8) + " ";
    s += Environment.NewLine;
  }
  return s;
}
```

You may want to parameterize the number of decimals to display, or the column-width padding, or both.

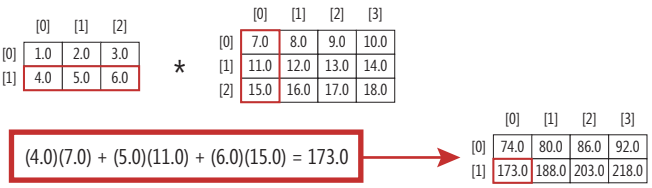


Figure 2 **Matrix Multiplication**

Matrix Multiplication

The Microsoft .NET Framework 4 and later provide a neat way to significantly improve the performance of a matrix multiplication method. Matrix multiplication is illustrated in **Figure 2**.

Note that the computation of each cell value of the result does not depend on any other cell values in the result, so each computation is independent and they can potentially be performed in parallel on a machine with multiple processors. Here's a standard approach to matrix multiplication:

```
static double[][] MatrixProduct(double[][] matrixA, double[][] matrixB)
{
  int aRows = matrixA.Length; int aCols = matrixA[0].Length;
  int bRows = matrixB.Length; int bCols = matrixB[0].Length;
  if (aCols != bRows)
    throw new Exception("Non-conformable matrices in MatrixProduct");

  double[][] result = MatrixCreate(aRows, bCols);

  for (int i = 0; i < aRows; ++i) // each row of A
    for (int j = 0; j < bCols; ++j) // each col of B
      for (int k = 0; k < aCols; ++k)
        result[i][j] += matrixA[i][k] * matrixB[k][j];

  return result;
}
```

The Task Parallel Library makes it easy to code a simple parallelized version of matrix multiplication.

Because matrix multiplication is an O(n^3) operation, performance can be an issue. For example, if matrix A has size 300 x 200 and matrix B has size 200 x 400, computing the product of A and B requires 300 * 200 * 400 = 24,000,000 multiplications. The Task Parallel Library (TPL) in the System.Threading.Tasks namespace in the .NET Framework 4 and later makes it easy to code a simple parallelized version of matrix multiplication. One possibility is:

```
static double[][] MatrixProduct(double[][] matrixA, double[][] matrixB)
{
  // error check, compute aRows, aCols, bCols
  double[][] result = MatrixCreate(aRows, bCols);

  Parallel.For(0, aRows, i =>
    {
      for (int j = 0; j < bCols; ++j)
        for (int k = 0; k < aCols; ++k)
          result[i][j] += matrixA[i][k] * matrixB[k][j];
    }
  );

  return result;
}
```

This version chops up the computations by rows. Behind the scenes, the TPL generates all the complex synchronization plumbing code to perform the computations on multiple processors.

Consistency Testing

An interesting aspect of a library of methods that are related to each other is that it's often possible to test them by checking to see if they produce consistent results. For example, suppose you have a method that creates a random matrix:

```
static double[][] MatrixRandom(int rows, int cols,
  double minVal, double maxVal, int seed)
{
  // return matrix with values between minVal and maxVal
  Random ran = new Random(seed);
  double[][] result = MatrixCreate(rows, cols);
  for (int i = 0; i < rows; ++i)
    for (int j = 0; j < cols; ++j)
      result[i][j] = (maxVal - minVal) * ran.NextDouble() + minVal;
  return result;
}
```

Additionally, suppose you have a matrix that creates the identity matrix—that is, a square matrix with 1.0s on the main diagonal, 0.0s elsewhere:

```
static double[][] MatrixIdentity(int n)
{
  double[][] result = MatrixCreate(n, n);
  for (int i = 0; i < n; ++i)
    result[i][i] = 1.0;
  return result;
}
```

And suppose you have a method that compares two matrices for equality:

```
static bool MatrixAreEqual(double[][] matrixA,
  double[][] matrixB, double epsilon)
{
  // true if all values in A == corresponding values in B
  int aRows = matrixA.Length;
  int bCols = matrixB[0].Length;
  for (int i = 0; i < aRows; ++i) // each row of A and B
    for (int j = 0; j < aCols; ++j) // each col of A and B
      if (Math.Abs(matrixA[i][j] - matrixB[i][j]) > epsilon)
        return false;
  return true;
 }
```

Notice the MatrixAreEqual method does not compare cell values for exact equality because the values are type double. Instead, the method checks to see if all the cell values are very close (within epsilon) to each other.

Because the product of any square matrix m and the identity matrix of the same dimension is equal to the original matrix m, you can test the matrix product method along the lines of this code:

```
double[][] m = MatrixRandom(4, 4, -9.0, 9.0, 0);
double[][] i = MatrixIdentity(4);
double[][] mi = MatrixProduct(m, i);
if (MatrixAreEqual(m, mi, 0.00000001) == true)
  Console.WriteLine("Consistent result");
else
  Console.WriteLine("Something is wrong");
```

Consistency checking lends itself well to random input testing.

Matrix Decomposition

Matrix decomposition takes a square matrix M and computes two new square matrices that when multiplied together give the original matrix M. The idea is similar to ordinary number factoring: the number 6 can be factored into 2 and 3 because 2 * 3 = 6. At first it may seem like there's little point in decomposing a matrix, but it turns out that matrix decomposition makes the very difficult task

Figure 3 **A Matrix Decomposition Method**

```
static double[][] MatrixDecompose(double[][] matrix,
  out int[] perm, out int toggle)
{
  // Doolittle LUP decomposition.
  // assumes matrix is square.

  int n = matrix.Length; // convenience
  double[][] result = MatrixDuplicate(matrix);
  perm = new int[n];
  for (int i = 0; i < n; ++i) { perm[i] = i; }

  toggle = 1;

  for (int j = 0; j < n - 1; ++j) // each column
  {
    double colMax = Math.Abs(result[j][j]); // largest val in col j
    int pRow = j;
    for (int i = j + 1; i < n; ++i)
    {
      if (result[i][j] > colMax)
      {
        colMax = result[i][j];
        pRow = i;
      }
    }

    if (pRow != j) // swap rows
    {
      double[] rowPtr = result[pRow];
      result[pRow] = result[j];
      result[j] = rowPtr;

      int tmp = perm[pRow]; // and swap perm info
      perm[pRow] = perm[j];
      perm[j] = tmp;

      toggle = -toggle; // row-swap toggle
    }

    if (Math.Abs(result[j][j]) < 1.0E-20)
      return null; // consider a throw

    for (int i = j + 1; i < n; ++i)
    {
      result[i][j] /= result[j][j];
      for (int k = j + 1; k < n; ++k)
        result[i][k] -= result[i][j] * result[j][k];
    }
  } // main j column loop
  return result;
}
```

Figure 4 **The HelperSolve Method**

```
static double[] HelperSolve(double[][] luMatrix, double[] b)
{
  // solve luMatrix * x = b
  int n = luMatrix.Length;
  double[] x = new double[n];
  b.CopyTo(x, 0);

  for (int i = 1; i < n; ++i)
  {
    double sum = x[i];
    for (int j = 0; j < i; ++j)
      sum -= luMatrix[i][j] * x[j];
    x[i] = sum;
  }

  x[n - 1] /= luMatrix[n - 1][n - 1];
  for (int i = n - 2; i >= 0; --i)
  {
    double sum = x[i];
    for (int j = i + 1; j < n; ++j)
      sum -= luMatrix[i][j] * x[j];
    x[i] = sum / luMatrix[i][i];
  }
  return x;
}
```

of matrix inversion quite a bit simpler. There are many different kinds of matrix decomposition, and each kind can be computed using several different algorithms. The technique presented in this article is called LUP decomposition and uses Doolittle's method with partial pivoting.

To understand LUP decomposition, it's helpful to first understand the simpler LU decomposition, which was introduced by the famous mathematician Alan Turing. Suppose you have this 4 x 4 matrix M:

```
 9.000   5.000   3.000   4.000
 4.000   8.000   2.000   5.000
 3.000   5.000   7.000   1.000
 2.000   6.000   0.000   8.000
```

One possible LU decomposition of M is L =

```
 1.000   0.000   0.000   0.000
 0.444   1.000   0.000   0.000
 0.333   0.577   1.000   0.000
 0.222   0.846  -0.219   1.000
```

And U =

```
 9.000   5.000   3.000   4.000
 0.000   5.778   0.667   3.222
 0.000   0.000   5.615  -2.192
 0.000   0.000   0.000   3.904
```

This works because L * U = M. Notice that the lower L matrix has 1.0s on the diagonal and 0.0s in the upper right. In other words, the significant cell values of the lower matrix are in the lower left. Similarly the significant cell values of the upper matrix are on the main diagonal and in the upper right.

Notice as well that there's no overlap of the locations of the significant cell values in L and U. So, instead of generating two results matrices, L and U, matrix decomposition usually stores both the lower and upper results into a single matrix that holds both L and U to save memory space.

LUP matrix decomposition is a slight but important variation on LU decomposition. LUP decomposition takes a matrix M and produces L and U matrices but also a P array. The product of L and U in LUP is not exactly the original matrix M but instead is a version of M where some of the rows have been rearranged. The way in which the rows have been rearranged is stored into the P array; this information can be used to reconstruct the original matrix M.

A close cousin to the Doolittle decomposition presented in this article is called Crout's decomposition. The main difference between Doolittle and Crout is that Doolittle places 1.0s on the main diagonal of the L matrix and Crout places 1.0s on the main diagonal of the U matrix.

The reason LUP decomposition is used more often than LU decomposition is subtle. As you'll see shortly, matrix decomposition is used to compute the inverse of a matrix. However, when matrix decomposition is used as a helper for matrix inversion, it turns out that the inversion will fail if there's a 0.0 value on the main diagonal of the LU matrix. So in LUP decomposition, when a 0.0 value ends up on the main diagonal, the algorithm exchanges two rows to move the 0.0 value off the diagonal and keeps track of which rows were exchanged in the P array.

Figure 3 lists a matrix decomposition method.

The method could be called like this:

```
double[][] m = MatrixRandom(4, 4, -9.0, 9.0, 0);
int[] perm;
int toggle;
double luMatrix = MatrixDecompose(m, out perm, out toggle);
```

The MatrixDecompose method accepts as its input a square matrix. The method has three return values. The explicit return is a permuted LU matrix. The method returns two values as out parameters. One is a permutation array that holds information about how the rows were permuted. The second out parameter is a toggle value that's either +1 or -1 depending on whether the number of row exchanges was even (+1) or odd (-1). The toggle value is not used for matrix inversion but is needed if matrix decomposition is used to compute the determinant of a matrix.

The MatrixDecompose method is fairly tricky, but realistically, there are only a few details you need to understand to modify the code. The version presented here allocates new memory for the LU return matrix using a helper method MatrixDuplicate:

```
static double[][] MatrixDuplicate(double[][] matrix)
{
  // assumes matrix is not null.
  double[][] result = MatrixCreate(matrix.Length, matrix[0].Length);
  for (int i = 0; i < matrix.Length; ++i) // copy the values
    for (int j = 0; j < matrix[i].Length; ++j)
      result[i][j] = matrix[i][j];
  return result;
}
```

An alternative approach is to compute the result into the input matrix in order to save memory. With C# semantics, this would make the matrix parameter a ref parameter because it's used for both input and output. Using this approach, the method signature would be:

```
static void MatrixDecompose(ref double[][] matrix, out int[] perm,
  out int toggle)
```

Or, because the explicit return value has been eliminated, you could use it for the permutation array or the exchanges toggle. For example:

```
static int[] MatrixDecompose(ref double[][] matrix, out int toggle)
```

You might want to eliminate the toggle parameter to simplify the method signature if you don't intend to use matrix decomposition to compute a determinant.

Another area of MatrixDecompose you might want to modify is this statement:

```
if (Math.Abs(result[j][j]) < 1.0E-20)
  return null;
```

In words, this code means: "If, even after exchanging two rows because there was a 0.0 value on the main diagonal, there's still a 0.0 there, then return null." You may want to modify the arbitrary epsilon value for the equality of zero check from 1.0E-20 to some other value based on the precision characteristics of your system. And instead of returning null, you might want to throw an exception; if the method were to be called as part of matrix inversion, the inversion would fail. Finally, if you're using matrix decomposition for some purpose other than matrix inversion, you may want to eliminate this statement altogether.

Matrix Inversion

The key to using matrix decomposition to invert a matrix is to write a helper method that solves a system of equations. This key helper method is presented in **Figure 4**.

The HelperSolve method finds an array x that when multiplied by an LU matrix gives array b. The method is quite clever, and you can only fully understand it by tracing through a few examples. There are two loops. The first loop uses forward substitution on the lower part of the LU matrix. The second loop uses backward substitution on the upper part of the LU matrix. Some different matrix decomposition implementations call their analogous method something like luBackSub.

Although the code is short, it's tricky, but there shouldn't be any scenarios where you'd need to modify the code. Notice that HelperSolve accepts an LU matrix from MatrixDecompose but doesn't use the perm out-parameter. This implies HelperSolve is in fact a helper method and needs additional wrapper code to solve a system of equations. If you refactor the code in this article to an OOP design, you might want to make HelperSolve a private method.

With the HelperSolve method in place, the matrix inversion method can be implemented, as shown in **Figure 5**.

Again, the code is tricky. The inversion algorithm is based on the idea that the product of a matrix M and its inverse is the identity matrix. Method MatrixInverse essentially solves a system of equations Ax = b where A is an LU matrix decomposition and the b constants are either 1.0 or 0.0 and correspond to the identity matrix. Notice that MatrixInverse uses the perm array from the call to MatrixDecompose.

Calling the MatrixInverse method could look like this:

```
double[][] m = MatrixRandom(4, 4, -9.0, 9.0, 0);
double[][] inverse = MatrixInverse(m);
Console.WriteLine(MatrixAsString(inverse));
```

To summarize, an important matrix operation is matrix inversion, which is quite difficult. One approach is to decompose the original matrix, write a helper solve method that performs forward and backward substitution, and then use the decomposition together with the LU permutation array and the helper solve method to find the inverse. This approach may seem complicated, but it's usually more efficient and easier than computing a matrix inverse directly.

Matrix Determinant

With a matrix decomposition method in hand, it's easy to code a method to compute the determinant of a matrix:

```
static double MatrixDeterminant(double[][] matrix)
{
  int[] perm;
  int toggle;
  double[][] lum = MatrixDecompose(matrix, out perm, out toggle);
  if (lum == null)
    throw new Exception("Unable to compute MatrixDeterminant");
  double result = toggle;
  for (int i = 0; i < lum.Length; ++i)
    result *= lum[i][i];
  return result;
}
```

As it turns out, somewhat surprisingly, the determinant of a matrix is just plus or minus (depending on the toggle value) the product of the values on the main diagonal of the matrix LUP decomposition. Notice that there's an implicit type conversion of the value of toggle from int to double. In addition to adding error checking to MatrixDeterminant, you might want to add a short-circuit return in situations where the input matrix has size 1 (then return the single value) or size 2 x 2 (then return a*d - b*c). Calling the determinant method could look like this:

```
double[][] m = MatrixRandom(4, 4, -9.0, 9.0, 0);
double det = MatrixDeterminant(m);
Console.WriteLine("Determinant = " + det.ToString("F2"));
```

Figure 5 **The MatrixInverse Method**

```
static double[][] MatrixInverse(double[][] matrix)
{
  int n = matrix.Length;
  double[][] result = MatrixDuplicate(matrix);

  int[] perm;
  int toggle;
  double[][] lum = MatrixDecompose(matrix, out perm, out toggle);
  if (lum == null)
    throw new Exception("Unable to compute inverse");

  double[] b = new double[n];
  for (int i = 0; i < n; ++i)
  {
    for (int j = 0; j < n; ++j)
    {
      if (i == perm[j])
        b[j] = 1.0;
      else
        b[j] = 0.0;
    }

    double[] x = HelperSolve(lum, b);

    for (int j = 0; j < n; ++j)
      result[j][i] = x[j];
  }
  return result;
}
```

Solving Systems of Equations

The HelperSolve method can be easily adapted to solve a system of linear equations:

```
static double[] SystemSolve(double[][] A, double[] b)
{
  // Solve Ax = b
  int n = A.Length;
  int[] perm;
  int toggle;
  double[][] luMatrix = MatrixDecompose(A, out perm, out toggle);
  if (luMatrix == null)
    return null; // or throw
  double[] bp = new double[b.Length];
  for (int i = 0; i < n; ++i)
    bp[i] = b[perm[i]];
  double[] x = HelperSolve(luMatrix, bp);
  return x;
}
```

Here's the code that produced the screenshot in **Figure 1** to solve the following system:

```
3x0 + 7x1 + 2x2 + 5x3 = 49
 x0 + 8x1 + 4x2 + 2x3 = 30
2x0 +  x1 + 9x2 + 3x3 = 43
5x0 + 4x1 + 7x2 +  x3 = 52

double[][] m = MatrixCreate(4, 4);
m[0][0] = 3.0; m[0][1] = 7.0; m[0][2] = 2.0; m[0][3] = 5.0;
m[1][0] = 1.0; m[1][1] = 8.0; m[1][2] = 4.0; m[1][3] = 2.0;
m[2][0] = 2.0; m[2][1] = 1.0; m[2][2] = 9.0; m[2][3] = 3.0;
m[3][0] = 5.0; m[3][1] = 4.0; m[3][2] = 7.0; m[3][3] = 1.0;

double[] b = new double[] { 49.0, 30.0, 43.0, 52.0 };
double[] x = SystemSolve(m, b);
Console.WriteLine("\nSolution is x = \n" + VectorAsString(x));
```

Notice that SystemSolve rearranges its b input parameter using the perm array from MatrixDecompose before calling HelperSolve.

Understanding the Permutation Array

The last few lines of output in the screenshot in **Figure 1** indicate that it's possible to multiply the L and U matrices in such a way as to get the original matrix. Knowing how to do this will not enable you to solve practical matrix problems, but it will help you understand the P part of LUP decomposition. Regenerating an original matrix from its L and U components can also be useful for testing your matrix library methods for consistency.

One way to regenerate an original matrix after LUP decomposition is to multiply L and U, and then permute the rows of the product using the P array:

```
// create matrix m
// call MatrixDecompose, returning lu and perm
// extract the lower part of lu as matrix 'lower'
// extract the upper part of lu as matrix 'upper'
double[][] lu = MatrixProduct(lower, upper);
double[][] orig = UnPermute(lu, perm);
if (MatrixAreEqual(orig, m, 0.000000001) == true)
  Console.WriteLine("L and U unpermuted using perm array");
```

The UnPermute method can be coded like this:

```
static double[][] UnPermute(double[][] luProduct, int[] perm)
{
  double[][] result = MatrixDuplicate(luProduct);
  int[] unperm = new int[perm.Length];
  for (int i = 0; i < perm.Length; ++i)
    unperm[perm[i]] = i; // create un-perm array
  for (int r = 0; r < luProduct.Length; ++r) // each row
    result[r] = luProduct[unperm[r]];
  return result;
}
```

A second approach for regenerating an original matrix from its LUP decomposition is to convert the perm array to a perm matrix and then multiply the perm matrix and the combined LU matrix:

```
// create matrix m
// call MatrixDecompose, returning lu and perm
// extract the lower part of lu as matrix 'lower'
// extract the upper part of lu as matrix 'upper'
double[][] permMatrix = PermArrayToMatrix(perm);
double[][] orig = MatrixProduct(permMatrix, lu);
if (MatrixAreEqual(orig, m, 0.000000001) == true)
  Console.WriteLine("L and U unpermuted using perm matrix");
```

A perm matrix is a square matrix with one 1.0 value in each row and each column. The method that creates a perm matrix from a perm array can be coded like so:

```
static double[][] PermArrayToMatrix(int[] perm)
{
  // Doolittle perm array to corresponding matrix
  int n = perm.Length;
  double[][] result = MatrixCreate(n, n);
  for (int i = 0; i < n; ++i)
    result[i][perm[i]] = 1.0;
  return result;
}
```

Wrapping Up

There are many algorithms that require solving a system of linear equations, finding the inverse of a matrix or finding the determinant of a matrix. Using matrix decomposition is an effective technique for performing all these operations. The code presented here can be used in situations where you want no external dependencies in your code base or you need the ability to customize the operations in order to improve performance or modify functionality. Take the red pill! ■

Dr. James McCaffrey *works for Volt Information Sciences Inc., where he manages technical training for software engineers working at the Microsoft Redmond, Wash., campus. He has worked on several Microsoft products including Internet Explorer and MSN Search. McCaffrey is the author of .NET Test Automation Recipes (Apress, 2006). He can be reached at jammc@microsoft.com.*

Thanks to the following technical experts for reviewing this article: *Paul Koch and Dan Liebling*

JAMES MCCAFFREY

Graph-Based Shortest-Path Analysis with SQL

In this column I explain how to implement a graph-based shortest-path analysis in situations where you have graph data stored in a SQL database and want to use native T-SQL language processing. Traditional shortest-path approaches assume that the graph representation is stored in a data structure in machine memory. But large graphs, such as those that represent social networks, often won't fit into memory, so storing and processing them using SQL is one option. The best way to understand where I'm headed is to look at the example graph in **Figure 1**, and the screenshot of a demo run in **Figure 2**.

The graph shown in **Figure 1** is artificially small and has eight nodes (sometimes called vertices or vertexes) with IDs 111 through 888. You can imagine that the nodes represent people or computers. The directional arrows (usually called edges) between nodes are relationships. You can imagine that an arrow between two nodes represents an exchange of e-mails. In this example the graph edges have weights indicated by the values 1.0 or 2.0. Edge weights can have different meanings depending on your particular problem scenario. For example, the weights can represent some measure of social affinity or an indicator of when a message was sent.

The term "shortest path" can have different meanings. Suppose you're interested in the shortest path between user 222 and user 444. This could mean the smallest number of hops to get from 222 to 444, which would be 4 (222 to 333 to 666 to 777 to 444). Or, shortest path could mean the smallest sum of weights between 222 and 444, which would be 5.0 (1.0 + 1.0 + 1.0 + 2.0).

In the left window in **Figure 2**, you can see I'm working with a database called dbShortPathDemo in SQL Server Management Studio. In the upper-right window I have a T-SQL script named ShortestPath.sql that defines and populates the demo database with information that corresponds to the graph in **Figure 1**, and code that defines a stored procedure named usp_ShortestPath. The stored procedure has just been called to analyze the shortest path between user 222 and user 444. In the bottom-right window you can see the results. The shortest path is given by the string "444;777;666;333;222." The shortest path in terms of weight is 5.0 (displayed without the decimal). The bottom-right part of the image shows the final state of a table named tblAlgorithmData, which is the key to implementing the shortest-path code.

Code download available at archive.msdn.microsoft.com/mag201212TestRun.

In the sections that follow, I'll walk you through the T-SQL code that generated the screenshot in **Figure 2** so you'll be able to adapt the code to meet your own problem scenarios. I'm assuming you're primarily a non-SQL developer (meaning you most often use C# or some other procedural programming language) but have some basic SQL knowledge. I present all the T-SQL code explained in this article; the code is also available at archive.msdn.microsoft.com/mag201211TestRun.

> Traditional shortest-path approaches assume that the graph representation is stored in a data structure in machine memory.

Setting up the Demo Database

To create a demo database I launched SQL Server Management Studio and connected to my server machine. I used SQL Server 2008, but the code presented here, with minor modifications, should work with most earlier and all newer versions of SQL Server. After connecting, I clicked File | New Query to get an editing window, then typed T-SQL code to create my dummy database:

```
use master
go

if exists(select * from sys.sysdatabases where name='dbShortPathDemo')
drop database dbShortPathDemo
go

create database dbShortPathDemo
go

use dbShortPathDemo
go
```

Here I used all the default parameter values for the create database statement. In many scenarios you'll be working with an existing database with real data, but I recommend experimenting with a small demo database. I prefer to use lowercase T-SQL code for the most part, which might offend SQL purists. I used the mouse to select these nine lines of code and then hit the F5 key to execute them. After verifying the demo database was created, I saved the script as ShortestPath.sql.

Next, I created a table within the demo database to hold the data that defines the graph to be analyzed:

```
create table tblGraph
(
fromNode bigint not null,
toNode bigint not null,
edgeWeight real not null
)
go
```

I used the mouse to highlight just these seven lines of code (so I wouldn't re-execute the previous lines) and hit F5 to create the table. I use type bigint for node IDs. Other common node ID types you might run into include int for a relatively simple employee ID and varchar(11) for a Social Security number in xxx-xx-xxxx format. I use type real for column edgeWeight. T-SQL type real is just a convenient alias for float(24), which is similar to C# type double. In many scenarios you might not have an explicit edge weight between nodes, and you can omit the edgeWeight column.

Next, I populated table tblGraph by entering, highlighting and executing these 15 lines of code:

```
insert into tblGraph values(111,222,1.0)
insert into tblGraph values(111,555,1.0)
insert into tblGraph values(222,111,2.0)
insert into tblGraph values(222,333,1.0)
insert into tblGraph values(222,555,1.0)
insert into tblGraph values(333,666,1.0)
insert into tblGraph values(333,888,1.0)
insert into tblGraph values(444,888,1.0)
insert into tblGraph values(555,111,2.0)
insert into tblGraph values(555,666,1.0)
insert into tblGraph values(666,333,2.0)
insert into tblGraph values(666,777,1.0)
insert into tblGraph values(777,444,2.0)
insert into tblGraph values(777,888,1.0)
go
```

If you refer to **Figure 1**, you'll see that each insert statement corresponds to one of the arrows in the graph.

Next, I created indexes on the fromNode and toNode columns in tblGraph to improve performance when the shortest-path stored procedure accesses the graph data:

```
create nonclustered index idxTblGraphFromNode on tblGraph(fromNode)
go
create nonclustered index idxTblGraphToNode on tblGraph(toNode)
go
```

I then created and populated a table to hold a list of unique node IDs:

```
create table tblNodes
(
nodeID bigint primary key
)
go

insert into tblNodes
select distinct fromNode from tblGraph
union
select distinct toNode from tblGraph
go
```

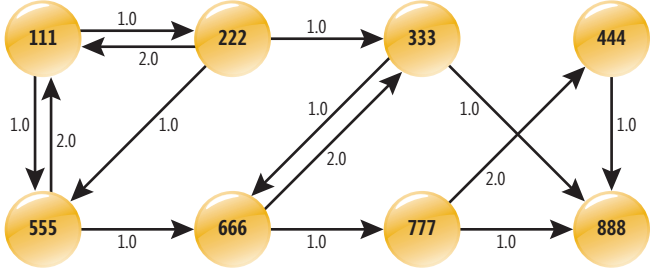


Figure 1 **Demo Graph for Shortest-Path Analysis**

A table of just node IDs isn't required for shortest-path analysis, but it's useful if you want to perform error checking on the start node and end node input parameters to the shortest-path stored procedure. By applying the primary key clause to the nodeID column, I implicitly create a clustered index on that column. The SQL union statement, used with two or more select-distinct statements, is a convenient way to fetch a list of unique values from a table.

> Large graphs, such as those that represent social networks, often won't fit into memory, so storing and processing them using SQL is one option.

The Algorithm Data Table

The key to understanding and modifying the shortest-path stored procedure presented in this article is understanding a table of information named tblAlgorithmData. You can create the table by executing these T-SQL statements:

```
create table tblAlgorithmData
(
nodeID bigint not null,
dist real not null,
pred bigint not null,
inQueue bit not null
)
go

create index idxTblAlgorithmDataNodeID on tblAlgorithmData(nodeID)
create index idxTblAlgorithmDataDist on tblAlgorithmData(dist)
create index idxTblAlgorithmDataPred on tblAlgorithmData(pred)
create index idxTblAlgorithmDataInQueue on tblAlgorithmData(inQueue)
go
```

The table will have (at most) one row for each unique node in the graph, so for the example in this article, the table will have (at most) eight rows. Because nodeID is a unique identifier, I could've defined it as a primary key column. The dist column is the current distance from the start node to the node that has nodeID. This dist value changes as the stored procedure executes but holds the final distance when the stored procedure finishes. If you look at **Figure 2**, you can see that the dist value for node 444, the end node, is 5.0 units when the stored procedure finished.

The pred column holds the immediate predecessor to the node with nodeID of the shortest path from parameter start node to end node. For example, in **Figure 2**, the end node is 444. Its predecessor is 777. The predecessor of 777 is 666. The predecessor of 666 is 333. The predecessor of 333 is 222, the start node. Putting these pred values together gives the shortest path from end node to start node: 444 to 777 to 666 to 333 to 222. Notice that the algorithm I use gives just one shortest path, even in situations where there are two or more paths with the same shortest total distance; in this example, the path 444 to 777 to 666 to 555 to 222 also has a total distance of 5.0 units.

The inQueue column holds a bit value (functionally similar to C# type Boolean) that indicates whether the associated node should be reexamined as part of the shortest path. Put another way, if the value of inQueue is 1, the associated node needs to be examined by the algorithm as a neighbor to the current node in the algorithm. If the value of inQueue is 0, the associated node doesn't need to be examined as a neighbor. The column name inQueue is somewhat misleading because the algorithm doesn't really use an explicit queue, so you might want to consider using a name such as isActive. I use the name inQueue because in procedural programming language implementations of shortest-path algorithms, an explicit queue is often used and so the name is somewhat traditional.

The Algorithm

The algorithm I present in this article is a variation of Dijkstra's Shortest Path (DSP) algorithm. In non-SQL pseudo-code, the variation of the DSP algorithm I use is presented in **Figure 3**.

The DSP algorithm is one of the most famous algorithms in computer science and you can find many painfully detailed explanations on the Internet, but very few complete implementations that use SQL. Although short, the DSP algorithm is very tricky, and the only way I was able to fully understand it was to trace through several examples using paper and pencil. The essence of the algorithm is that at a given node u, you'll know the current shortest distance from the start node to u. Then you find all neighbors of u. For each neighbor node, v, you know the current distance from the start node to v. You can look up the distance from u to v. If the distance from start to u plus the distance from u to v is shorter than the current distance from start to v, you know you've found a shorter path from start to v. The inQueue variable prevents the algorithm from revisiting a node once it's known that revisiting that node will not find a shorter path.

The Stored Procedure

I implemented the shortest path as a T-SQL stored procedure. The stored procedure definition begins:

```
create procedure usp_ShortestPath
  @startNode bigint,
  @endNode bigint,
  @path varchar(5000) output,
  @totDist real output
as
begin
  ...
```

I prepend the stored procedure usp_ShortestPath name with "usp" (user-defined stored procedure) to distinguish it from system stored procedures ("sp"), extended stored procedures ("xp") and CLR stored procedures ("csp"). Parameters @startNode and @endNode are input parameters. The stored procedure has two result output parameters, @path and @totDist. The @path is arbitrarily set to type varchar(5000)—you might need to increase the 5000 maximum size depending on the type of node ID you're using and the maximum path you expect.

Next, I initialize table tblAlgorithmData with information for the start node:

```
truncate table tblAlgorithmData
insert into tblAlgorithmData
values (@startNode, 0.0, -1, 1)
...
```

The truncate statement erases the contents of tblAlgorithmData from any previous call to usp_ShortestPath. Recall the columns for tblAlgorithmData are nodeID, dist, pred and inQueue. The distance from the start node to itself is 0.0, the predecessor of the start node is set to -1 to indicate undefined, and the inQueue bit is set to 1 to indicate true.

This code assumes that tblAlgorithmData has already been created in the current database as a permanent table. In some

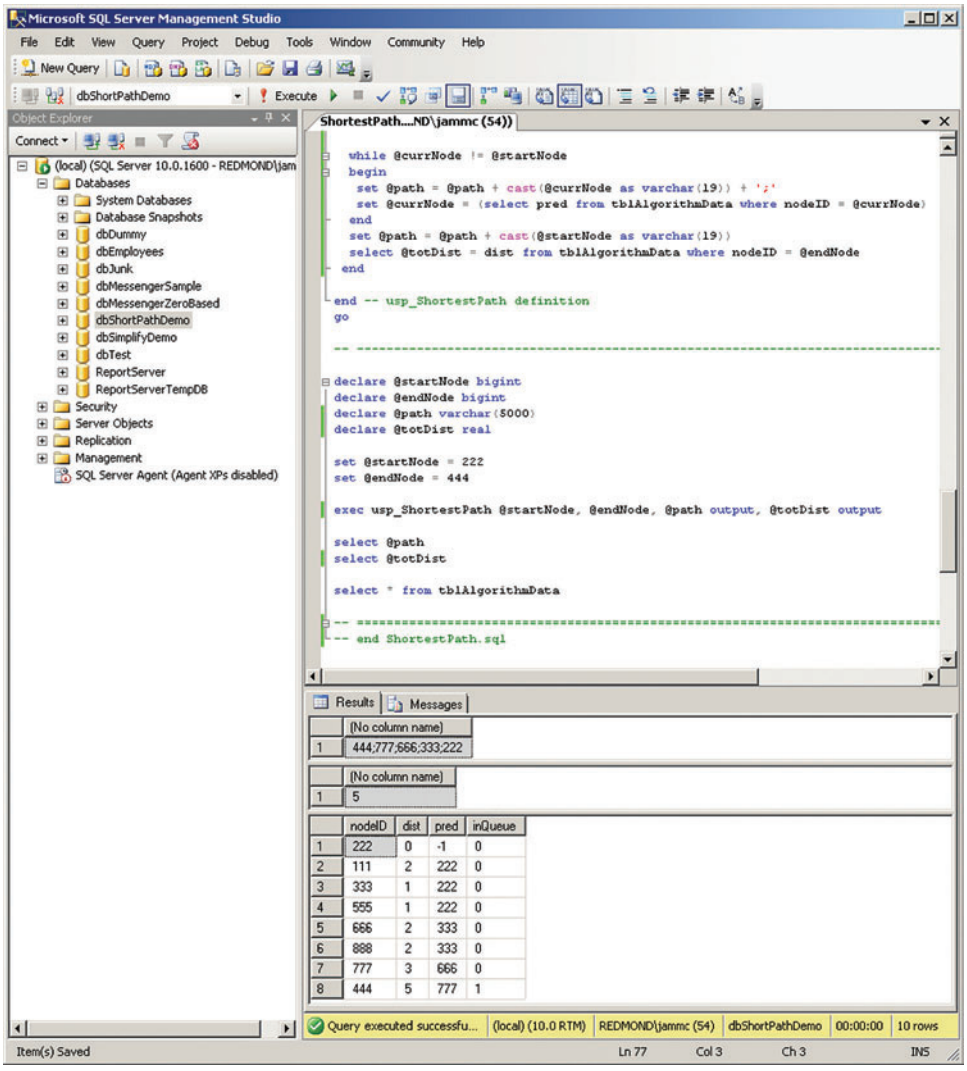


Figure 2 **Shortest-Path Analysis with T-SQL**

situations you might not have the security permissions needed to create the table; in these situations you can either ask the appropriate database administrator to do so for you, or you can create tblAgorithmData inside the stored procedure as a temp table or possibly a table variable.

This code also assumes that the values of both @startNode and @endNode are valid. If you have a table of all node IDs, you could check for this along the lines of:

```
if not exists(select nodeID from tblNodes where @startNode = nodeID)
  // Error out in some way //
```

Next, I prepare the main processing loop:

```
declare @u bigint
declare @d real = 0.0
declare @done bit = 0

while @done = 0
begin
...
```

Here @u is the node ID of the current node in the algorithm. Variable @d is a convenience and holds the distance from @startNode to node @u (as stored in tblAlgorithmData). The @done variable is a dummy loop control variable. You might want to put other explicit stopping criteria in the loop such as a maximum number of iterations, maximum hop count or maximum total path distance.

Inside the main processing loop, the first step is to find the value of @u—the node ID of the current node. This is the node that has the smallest dist column value in tblAlgorithmData and also has inQueue = 1:

```
select top 1 @u = nodeID, @d = dist from tblAlgorithmData
where inQueue = 1
order by dist asc
...
```

At this point the stored procedure checks for two loop exit conditions:

```
if @d = 2147483647.0 break
if @u = @endNode break
...
```

If the distance from the start node to @u (stored in @d) is equal to the somewhat mysterious-looking value of 2147483647.0, this means that the end node is not reachable from the start node. The value of 2147483647.0 is a stand-in for infinity. You can use any large value that can't actually appear as a distance. I picked 2147483647.0 because 2147483647 (without the decimal) is the largest value for SQL type int. The other break condition simply checks to see if the end node has been reached. Because of the way the algorithm searches the graph using a breadth-first approach, if the end node is hit, the shortest path has been found.

Next, the stored procedure checks each neighbor of the current node @u and if a neighbor hasn't been seen before, the neighbor is initialized into tblAlgorithmData:

```
insert into tblAlgorithmData
select toNode, 2147483647.0, -1, 1 from tblGraph where fromNode = @u
and not exists (select * from tblAlgorithmData where nodeID = toNode)
...
```

This SQL code is a bit subtle if you're a coder who rarely uses SQL. The insert statement is conditional upon the not exists statement, which can be interpreted as, "if the neighbor node ID (value toNode) is not already in tblAlgorithmData (as nodeID)." For each neighbor node where the conditional is true, the insert statement initializes tblAlgorithmData with the neighbor's nodeID, a dist value of infinity (as 2147483647.0), a predecessor of undefined (as -1), and an inQueue of true (as 1). Working with SQL statements that operate on sets of data, rather than iterating through a collection using a loop as you would with a procedural programming language, can be a difficult paradigm to master.

In this algorithm, nodes are initialized when they first appear as neighbor nodes. A significant alternative approach is to initialize all graph nodes at the very beginning of the algorithm. The problem with this approach is that for large graphs, populating tblAlgorithmData with possibly millions of nodes can take quite a bit of time.

At this point, the stored procedure marks the current node as processed:

```
update tblAlgorithmData set inQueue = 0
where nodeID = @u
...
```

Next, the stored procedure checks each neighbor to the current node to see if a new, shorter path to the neighbor has been found and, if so, updates the entries in tblAlgorithmData for that neighbor node:

```
  update tblAlgorithmData
  set dist = @d + tblGraph.edgeWeight, pred = @u
  from tblAlgorithmData inner join tblGraph on tblAlgorithmData.nodeID =
tblGraph.toNode
  where tblGraph.fromNode = @u and tblAlgorithmData.inQueue = 1
    and (@d + tblGraph.edgeWeight) < tblAlgorithmData.dist
end -- loop
...
```

This is by far the trickiest part of the shortest-path implementation. Table tblGraph is joined to tblAlgorithmData so that all data can be used in the update statement. The current node's ID is stored in @u and this value is matched to the fromNode in tblGraph. The neighbors to @u are stored in toNode in tblGraph, which is linked to the nodeID of tblAlgorithmData. Recall that @d holds the distance from the start node to the current node @u. The value in the edgeWeight column will hold the distance from @u to the neighbor.

Figure 3 **Shortest-Path Algorithm**

```
set dist of startNode to 0.0
set pred of startNode to undefined
set inQueue of startNode to true
while there are any nodes with inQueue = true
  set u = node with smallest dist and inQueue = true

  if dist of u is infinity break
  if u = endNode break

  fetch all neighbors of u
  foreach neighbor v that has inQueue = true
    if v has not been seen before then
      set dist of v to infinity
      set pred of v to undefined
      set inQueue of v to true
    end if
  end foreach

  set inQueue of u = false

  foreach neighbor v of u
    if ((dist from startNode to u) + (dist from u to v) <
      curr dist to v) then
        set dist of v to (dist from startNode to u) + (dist from u to v)
        set pred of v to u
    end if
  end foreach
end while

if (u != endNode) then
  path from startNode to endNode is unreachable
else
  retrieve path using pred values
  shortest path distance = dist of endNode
end if
```

The sum of these two distances is a possible new shorter path from the current distance from the start node to the neighbor, which is stored in column dist. If a new shorter distance has been found, the dist column is updated with that new shorter distance and the predecessor to the neighbor node is recorded as @u, the current node. In situations where you define shortest path to mean the fewest number of hops between start node and end node, you would replace dist = @d + tblGraph.edgeWeight with dist = @d + 1.

At this point the main processing loop has exited and the cause of the exit can be checked:

```
if (@u != @endNode)
  begin
    set @path = 'NOT REACHABLE'
    set @totDist = -1.0
  end
  else
  begin
    set @path = ''
    declare @currNode bigint
    set @currNode = @endNode
  ...
```

If the value in @u is the ID of the end node, the end node has been found; if @u is anything other than the ID of end node, the loop exited before finding the end node. In this case I set the path string to 'NOT REACHABLE' and assign an arbitrary shortest-path total distance of -1.0 to indicate not reachable. If the end node was in fact found, I initialize the shortest-path string to the empty string and set up local variable @currNode to iterate through tblAlgorithmData to construct the shortest-path string.

The stored procedure definition code concludes by constructing the shortest-path string and assigning the shortest-path total distance:

```
    ...
    while @currNode != @startNode
    begin
      set @path = @path + cast(@currNode as varchar(19)) + ';'
      set @currNode = (select pred from tblAlgorithmData where nodeID = @currNode)
    end

    set @path = @path + cast(@startNode as varchar(19))
    select @totDist = dist from tblAlgorithmData where nodeID = @endNode
  end -- else

end -- usp_ShortestPath definition
go
```

Here I use the T-SQL string concatenation "+" operator. I use varchar(19) because the largest possible value for my nodeID type of bigint is 9,223,372,036,854,775,807, which has 19 digits.

With the stored procedure created, I can find the shortest path between any two nodes; for example:

```
declare @startNode bigint
declare @endNode bigint
declare @path varchar(5000)
declare @totDist real

set @startNode = 222
set @endNode = 444

exec usp_ShortestPath @startNode, @endNode, @path output, @totDist output

select @path
select @totDist
```

Wrapping Up

Shortest-path graph analysis is likely to increase in importance as enterprises gather more data and store that data in a cloud environment. The code and explanation presented in this article should give you a solid basis for getting started with performing shortest-path analyses on your data. An important alternative to the native T-SQL language approach described in this article is to use a CLR stored procedure. Based on my experiences, a shortest-path analysis using a CLR stored procedure can give you improved performance at the expense of increased complexity. I'll present a CLR stored procedure approach to graph-based shortest-path analysis in a future article. ■

> If a new shorter distance has been found, the dist column is updated with that new shorter distance.

Dr. James McCaffrey *works for Volt Information Sciences Inc., where he manages technical training for software engineers working at the Microsoft Redmond, Wash., campus. He has worked on several Microsoft products including Internet Explorer and MSN Search. McCaffrey is the author of ".NET Test Automation Recipes" (Apress, 2006). He can be reached at jammc@microsoft.com.*

Thanks to the following technical expert for reviewing this article: *Shane Williams*

CHARLES PETZOLD

A Touch Interface for an Orienting Map

Whenever I'm a bit lost in a shopping mall or a museum, I search for a map, but at the same time I often feel some anxiety about what I'll find. I'm pretty sure the map will feature an arrow labeled, "You are here," but how will the map be oriented? If the map is mounted vertically, does the right side of the map actually correspond to my right and the bottom correspond to what's behind me? Or does the map need to be mentally removed from its mount and twisted in space to line up with the actual layout?

Maps that are mounted at an angle or parallel with the floor are much better—provided, that is, they're oriented correctly to begin with. Regardless of one's mental agility with spatial relations, maps are easier to read when they're parallel to the earth, or can be swiveled forward to align with the earth. Prior to the age of GPS, it was common to see people grappling with paper road maps by wildly twisting them to the right, left and upside down in search of the proper orientation.

Maps that are implemented in software on phones and other mobile devices have the potential to orient themselves based on a compass reading. This is the impetus behind my quest to display a map on a Windows Phone device that rotates relative to the phone. Such a map should be able to align itself with the surrounding landscape and potentially be more helpful to the lost among us.

Orienting the Map

My initial goal was a bit more ambitious than a rotating map. The program I envisioned would actually float a map in 3D space so it would always be parallel to the surface of the Earth, as well as being oriented with the compass.

A little experimentation convinced me that this approach was somewhat more extravagant than I needed. Although a tilted map with perspective is fine for GPS displays in automobiles, I think that's because the map is always tilted the same degree, and it somewhat mimics what you're seeing out the windshield. With a map on a mobile device, the tilting has the effect of compressing the map visuals without providing any additional information. A simple two-dimensional rotation seemed to be sufficient.

In my November column, I discussed how to use the Bing Maps SOAP Services to download and assemble 256-pixel square tiles into a map ("Assembling Bing Map Tiles on Windows Phone," msdn.microsoft.com/magazine/tktktktk). The tiles available from this Web service are organized into zoom levels, where each higher level has double the resolution of the previous level, which means that each tile covers the same area as four tiles in the next-higher level.

The program in last month's column contained application bar buttons labeled with plus and minus signs to increase and decrease the zoom level in discrete jumps. That type of interface is adequate for a map on a Web site, but for a phone, the only description that seems appropriate is "totally lame."

This means it's now time to implement a real touch interface that allows the map to be zoomed continuously.

Once I began adding that touch interface—a single finger to pan, two fingers to zoom in and out—I acquired a deep and enduring respect for the Maps application on Windows Phone, and for the Silverlight Map control. These maps obviously implement a much more sophisticated touch interface than what I've been able to manage.

For example, I don't think I've ever seen a black hole open up in the Maps app because a tile is missing. It's my experience that the screen is always entirely covered—although obviously sometimes with a tile that has been stretched beyond the point of recognition. Tiles are replaced with tiles of better resolution with a fade animation. Inertia is implemented in a very natural way, and the UI never gets jumpy while tiles are being downloaded.

My OrientingMap program (which you can download) comes nowhere close to the real Maps application. The panning and expansion is often jumpy, there's no inertia and blank areas frequently appear if tiles aren't downloaded quickly enough.

Despite these deficiencies, my program does succeed in maintaining an orientation of the map with the world it portrays.

Code download available at archive.msdn.microsoft.com/mag201212TouchAndGo.

Figure 1 **The MapTile.xaml File from OrientingMap**

```
<UserControl x:Class="OrientingMap.MapTile" ... >
  <Grid>
    <Image Stretch="None">
      <Image.Source>
        <BitmapImage x:Name="bitmapImage"
                     ImageOpened="OnBitmapImageOpened" />
      </Image.Source>
    </Image>

    <!-- Display quadkey for debugging purposes -->
    <TextBlock Name="txtblk"
               Visibility="Collapsed"
               Foreground="Red" />
  </Grid>

  <UserControl.RenderTransform>
    <ScaleTransform x:Name="scale" />
  </UserControl.RenderTransform>
</UserControl>
```

The Basic Issue

The Bing Maps SOAP Services give a program access to 256-pixel square map tiles from which it can construct larger composite maps. For road and aerial views, Bing Maps makes available 21 levels of zoom, where level 1 covers the Earth with four tiles, level 2 with 16 tiles, level 3 with 64 and so forth. Each level provides double the horizontal resolution and double the vertical resolution of the next lower level.

Tiles have a parent-child relationship: Except for tiles in level 21, every tile has four children in the next-higher level that together cover the same area as itself but with double the resolution.

When a program sticks to integral zoom levels—as the program presented in last month's column does—the individual tiles can be displayed in their actual pixel sizes. Last month's program always displays 25 tiles in a 5 × 5 array, for a total size of 1,280 pixels square. The program always positions this array of tiles so that the center of the screen corresponds to the phone's location on the map, which is a location somewhere in the center tile. Do the math and you'll find that even if a corner of the center tile sits in the center of the screen, this 1,280 pixel square size is adequate for the 480 × 800 screen size of the phone, regardless how it's rotated.

Because last month's program supports only discrete zoom levels and always centers the tiles based on the phone's location, it implements an extremely simplistic logic by completely replacing these 25 tiles whenever a change occurs. Fortunately, the download cache makes this process fairly fast if tiles are being replaced with previously downloaded tiles.

With a touch interface, this simple approach is no longer acceptable.

The hard part is definitely the scaling: For example, suppose the program begins by displaying map tiles of level 12 in their pixel sizes. Now the user puts two fingers on the screen and moves the fingers apart to expand the screen. The program must respond by scaling the tiles beyond their 256-pixel sizes. It can do this either with a ScaleTransform on the tiles themselves, or with a ScaleTransform applied to a Canvas on which the tiles are assembled.

But you don't want to scale these tiles indefinitely! At some point you want to replace each tile with four child tiles of the next-higher level and half the scaling factor. This replacement process would be fairly trivial if the child tiles were instantly available but, of course, they're not. They must be downloaded, which means that child tiles must be positioned visually on top of the parent, and only when all four child tiles have been downloaded can the parent be removed from the Canvas.

The opposite process must occur in a zoom out. As the user pinches two fingers together, the entire array of tiles can be scaled down, but at some point each group of four tiles should be replaced

Figure 2 **Much of the MainPage.xaml File for OrientingMap**

```
<phone:PhoneApplicationPage x:Class="OrientingMap.MainPage" ... >

  <Grid x:Name="LayoutRoot" Background="Transparent">

    <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12">
      <TextBlock Name="errorTextBlock"
                 HorizontalAlignment="Center"
                 VerticalAlignment="Top"
                 TextWrapping="Wrap" />

      <!-- Rotating Canvas with origin in center of screen -->
      <Canvas HorizontalAlignment="Center"
              VerticalAlignment="Center">

        <!-- Translating Canvas for panning -->
        <Canvas>

          <!-- Scaled Canvas for images -->
          <Canvas Name="imageCanvas"
                  HorizontalAlignment="Center"
                  VerticalAlignment="Center">
            <Canvas.RenderTransform>
              <ScaleTransform x:Name="imageCanvasScale" />
            </Canvas.RenderTransform>
          </Canvas>

          <!-- Circle to show location -->
          <Ellipse Name="locationDisplay"
                   Width="24"
                   Height="24"
                   Stroke="Red"
                   StrokeThickness="3"
                   HorizontalAlignment="Center"
                   VerticalAlignment="Center"
                   Visibility="Collapsed">
            <Ellipse.RenderTransform>
              <TranslateTransform x:Name="locationTranslate" />
            </Ellipse.RenderTransform>
          </Ellipse>

          <Canvas.RenderTransform>
            <TranslateTransform x:Name="imageCanvasTranslate" />
          </Canvas.RenderTransform>
        </Canvas>

        <Canvas.RenderTransform>
          <RotateTransform x:Name="imageCanvasRotate" />
        </Canvas.RenderTransform>
      </Canvas>

      <!-- Arrow to show north -->
      <Border HorizontalAlignment="Left"
              VerticalAlignment="Top"
              Background="Black"
              Width="36"
              Height="36"
              CornerRadius="18">
        <Path Stroke="White"
              StrokeThickness="3"
              Data="M 18 4 L 18 24 M 12 12 L 18 4 24 12">
          <Path.RenderTransform>
            <RotateTransform x:Name="northArrowRotate"
                             CenterX="18"
                             CenterY="18" />
          </Path.RenderTransform>
        </Path>
      </Border>

      <!-- "powered by bing" display -->
      <Border Background="Black"
              HorizontalAlignment="Center"
              VerticalAlignment="Bottom"
              CornerRadius="12"
              Padding="3">

        <StackPanel Name="poweredByDisplay"
                    Orientation="Horizontal"
                    Visibility="Collapsed">
          <TextBlock Text=" powered by "
                     Foreground="White"
                     VerticalAlignment="Center" />
          <Image Stretch="None">
            <Image.Source>
              <BitmapImage x:Name="poweredByBitmap" />
            </Image.Source>
          </Image>
        </StackPanel>
      </Border>
    </Grid>
  </Grid>
  ...
</phone:PhoneApplicationPage>
```

with a parent tile visually underneath the four tiles. Only when that parent tile has been downloaded can the four children be removed.

Additional Classes

As I discussed in last month's column, Bing Maps uses a numbering system called a "quadkey" to uniquely identify map tiles. A quadkey is a base-4 number: The number of digits in the quadkey indicates the zoom level, and the digits themselves encode an interleaved longitude and latitude.

To assist the OrientingMap program in working with quadkeys, the project includes a QuadKey class that defines properties to obtain parent and child quadkeys.

The OrientingMap project also has a new MapTile class that derives from UserControl. The XAML file for this control is shown in **Figure 1**. It has an Image element with its Source property set to a BitmapImage object for displaying the bitmap tile, as well as a ScaleTransform for scaling the entire tile up or down. (In practice, individual tiles are only scaled by positive and negative integral powers of 2.) For debugging, I put a TextBlock in the XAML file that displays the quadkey, and I've left that in: Simply change the Visibility attribute to Visible to see it.

The codebehind file for MapTile defines several handy properties: The QuadKey property allows the MapTile class itself to obtain the URI for accessing the map tile; a Scale property lets external code set the scaling factor; an IsImageOpened property indicates when the bitmap has been downloaded; and an ImageOpened property provides external access to the ImageOpened event of the BitmapImage object. These last two properties help the program determine when an image has been loaded so the program can remove any tiles that the image is replacing.

While developing this program, I initially pursued a scheme where each MapTile object would use its Scale property to determine when it should be replaced with a group of four child MapTile objects, or a parent MapTile. The MapTile itself would handle the creation and positioning of these new objects, setting handlers for the ImageOpened events, and would also be responsible for removing itself from the Canvas.

But I couldn't get this scheme to work very well. Consider an array of 25 map tiles that the user expands through the touch interface. These 25 tiles are replaced with 100 tiles, and then the 100 tiles are replaced with 400 tiles. Does this make sense? No, it doesn't, because the scaling has effectively moved many of these potential new tiles too far off the screen to be visible. Most of them shouldn't be created or downloaded at all!

Instead, I shifted this logic to MainPage. This class maintains a currentMapTiles field of type Dictionary<QuadKey, MapTile>. This stores all the MapTile objects currently on the display, even if they're still in the process of being downloaded. A method named RefreshDisplay uses the current location of the map and a scaling factor to assemble a validQuadKeys field of type List<QuadKey>. If a QuadKey object exists in validQuadKeys but not in currentMapTiles, a new MapTile is created and added to both the Canvas and currentMapTiles.

RefreshDisplay does not remove MapTile objects that are no longer needed, either because they've been panned off the screen or replaced with parents or children. That's the responsibility of a second important method named Cleanup. This method compares the validQuadKeys collection with currentMapTiles. If it finds an item in currentMapTiles that's not in validQuadKeys, it only removes that MapTile if validQuadKeys has no children, or if the children in validQuadKeys have all been downloaded, or if validQuadKeys contains a parent of that MapTile and that parent has been downloaded.

Making the RefreshDisplay and Cleanup methods more efficient—and invoking them less frequently—is one approach to improving the performance of OrientingMap.

Nested Canvases

The UI for the OrientingMap program requires two types of graphics transforms: translation for single-finger panning and scaling for two-finger pinch operations. In addition, orienting the map with the direction of north requires a rotation transform. To implement these with efficient Silverlight transforms, the MainPage.xaml file contains three levels of Canvas panels, as shown in **Figure 2**.

The Grid named ContentPanel contains the outermost Canvas as well as three elements that are always displayed in fixed locations on the screen: a TextBlock to report initialization errors, a Border containing a rotating arrow to display the direction of north and another Border to display the Bing logo.

The outermost Canvas has its HorizontalAlignment and VerticalAlignment properties set to Center, which shrinks the Canvas to a zero size positioned in the center of the Grid. The (0, 0) coordinate of this Canvas is therefore the center of the display. This centering is convenient for positioning tiles, and also allows scaling and rotation to occur around the origin.

The outermost Canvas is the one that's rotated based on the direction of north. Within this outermost Canvas is a second Canvas that has a TranslateTransform. This is for panning. Whenever a single finger sweeps across the screen, the entire map can be moved simply by setting the X and Y properties of this TranslateTransform.

Within this second Canvas is an Ellipse used to indicate the current location of the phone relative to the center of the map. When the user pans the map, this Ellipse moves as well. But if the phone's GPS reports a change in the location, a separate TranslateTransform on the Ellipse moves it relative to the map.

The innermost Canvas is named imageCanvas, and it's here that the map tiles are actually assembled. The ScaleTransform applied to this Canvas allows the program to increase or decrease this entire assemblage of map tiles based on the user zooming in or out with a pinch manipulation.

To accommodate the continuous zoom, the program maintains a zoomFactor field of type double. This zoomFactor has the same range as the tile levels—from 1 to 21—which means that it's actually the base-2 logarithm of the total map-scaling factor. Whenever the zoomFactor increases by 1, the scaling of the map doubles.

The first time the program is run, zoomFactor is initialized to 12, but the first time the user touches the screen with two fingers, it becomes a non-integral value and very likely remains a non-integral value thereafter. The program saves zoomFactor as a user setting and reloads it the next time the program is run. An initial integral baseLevel is calculated with a simple truncation:

```
baseLevel = (int)zoomFactor;
```

This baseLevel is always an integer in the range between 1 and 21, and hence it's directly suitable for retrieving tiles. From these two numbers, the program calculates a non-logarithmic scaling factor of type double:

```
canvasScale = Math.Pow(2, zoomFactor - baseLevel);
```

This is the scaling factor applied to the innermost Canvas. For example, if the zoomFactor is 10.5, then the baseLevel used to retrieve tiles is 10, and canvasScale is 1.414.

If the initial zoomFactor is 10.9, it might make more sense to set baseLevel at 11 and canvasZoom at 0.933. The program doesn't do that, but it's obviously a possible refinement.

Figure 3 **Touch Processing in OrientingMap**

```
void OnCompositionTargetRendering(object sender, EventArgs args)
{
  while (TouchPanel.IsGestureAvailable)
  {
    GestureSample gesture = TouchPanel.ReadGesture();

    switch (gesture.GestureType)
    {
      case GestureType.FreeDrag:
        // Adjust delta for rotation of canvas
        Vector2 delta = TransformGestureToMap(gesture.Delta);

        // Translate the canvas
        imageCanvasTranslate.X += delta.X;
        imageCanvasTranslate.Y += delta.Y;

        // Adjust the center longitude and latitude
        centerRelativeLongitude -= delta.X / (1 << baseLevel + 8) / canvasScale;
        centerRelativeLatitude -= delta.Y / (1 << baseLevel + 8) / canvasScale;

        // Accumulate the panning distance
        accumulatedDeltaX += delta.X;
        accumulatedDeltaY += delta.Y;

        // Check if that's sufficient to warrant a screen refresh
        if (Math.Abs(accumulatedDeltaX) > 256 ||
            Math.Abs(accumulatedDeltaY) > 256)
        {
          RefreshDisplay();
          accumulatedDeltaX = 0;
          accumulatedDeltaY = 0;
        }
        break;

      case GestureType.DragComplete:
        Cleanup();
        break;

      case GestureType.Pinch:
        // Get the old and new finger positions relative to canvas origin
        Vector2 newPoint1 = gesture.Position - canvasOrigin;
        Vector2 oldPoint1 = newPoint1 - gesture.Delta;

        Vector2 newPoint2 = gesture.Position2 - canvasOrigin;
        Vector2 oldPoint2 = newPoint2 - gesture.Delta2;

        // Rotate in accordance with the current rotation angle
        oldPoint1 = TransformGestureToMap(oldPoint1);
        newPoint1 = TransformGestureToMap(newPoint1);
        oldPoint2 = TransformGestureToMap(oldPoint2);
        newPoint2 = TransformGestureToMap(newPoint2);

        ...

        RefreshDisplay();
        break;

      case GestureType.PinchComplete:
        Cleanup();
        break;
    }
  }
}
```

One- and Two-Finger Touch Input

For touch input, I felt more comfortable using the XNA TouchPanel than the Silverlight Manipulation events. The MainPage constructor enables four types of XNA gestures: FreeDrag (panning), DragComplete, Pinch and PinchComplete. The TouchPanel is checked for input in a handler for the CompositionTarget.Rendering event, as shown in **Figure 3**. Due to its complexity, only a little of the Pinch processing is shown here.

The FreeDrag input is accompanied by Position and Delta values (both of type Vector2) indicating the current position of the finger, and how the finger has moved since the last TouchPanel event. The Pinch input supplements these with Position2 and Delta2 values for the second finger.

However, keep in mind that these Vector2 values are in-screen coordinates! Because the map is rotated relative to the screen—and the user expects the map to pan in the same direction as a finger moves—these values must be rotated based on the current map rotation, which occurs in a little method named TransformGestureToMap.

For FreeDrag processing, the delta value is then applied to the TranslateTransform in the XAML file, as well as two floating-point fields named centerRelativeLongitude and centerRelativeLatitude. These values range from 0 to 1 and indicate the longitude and latitude corresponding to the center of the screen.

At some point, the user might pan the map to a sufficient degree that new tiles need to be loaded. To avoid checking for that possibility with each touch event, the program maintains two fields named accumulatedDeltaX and accumulatedDeltaY, and only calls RefreshDisplay when either value goes above 256, which is the pixel size of the map tiles.

Because RefreshDisplay has a big job to do—determining what tiles should be visible on the screen based on centerRelativeLongitude and centerRelativeLatitude and the current canvasScale, and creating new tiles if necessary—it's best that it not be called for every change in touch input. One definite enhancement to the program would limit RefreshDisplay calls during Pinch input.

During touch processing, the Cleanup method is only called when the finger or fingers have left the screen. Cleanup is also called whenever a map tile has completed downloading.

The criteria for changing baseLevel—and thereby initiating a replacement of a parent map tile by children, or children by a parent—is very relaxed. The baseLevel is only incremented when canvasScale becomes greater than 2, and decremented when canvasScale drops to less than 0.5. Setting better transition points is another obvious enhancement.

The program now has only two application bar buttons: The first toggles between road and aerial view, and the second positions the map so that the current location is in the center.

Now I just need to figure out how to make the program help me navigate shopping malls and museums. ■

Charles Petzold *is a longtime contributor to* MSDN Magazine, *and the author of "Programming Windows, 6th edition" (O'Reilly Media, 2012), a book about writing applications for Windows 8. His Web site is charlespetzold.com.*

Thanks to the following technical expert for reviewing this article: *Thomas Petchel*

DAVID S. PLATT

Being Fully Digital

Microsoft is providing more UX guidance for developers with Windows 8 than it did with Windows Presentation Foundation and Silverlight, a change I welcome. One of the tenets of the new Windows UI style is to make your content fully digital. By this, Microsoft means that you should not spend screen space or CPU cycles or user attention on meatspace analogies, such as a book reader app displaying pages that look like physical paper. Apple in November announced it was moving in the same direction.

The design guidance makes sense. Mobile device screens are small compared to PCs; computing cycles are also more limited, and so are physical storage and battery power. A pixel spent on the deckle edge of a simulated book page means one less pixel for the book text. The page-flipping motion in a reader app is a PC indulgence, unaffordable on a mobile device.

Will users accept this? I think so. When personal computers first started catching on 30 years ago, our main UI idiom was making the computer display picture look like the physical thing it replaced. For example, the display of the Windows Cardfile program, shown in **Figure 1**, looked like an actual file of paper index cards (well, as close as we could get to it with the graphics of that time). But that changed as computer usage spread and evolved.

We've reached the point where informational content has been decoupled from its physical storage medium. Our computer representations no longer need to simulate their physical origins, such as page numbers or CD track numbers. The flexibility of digital presentation renders these useless at best, misleading at worst.

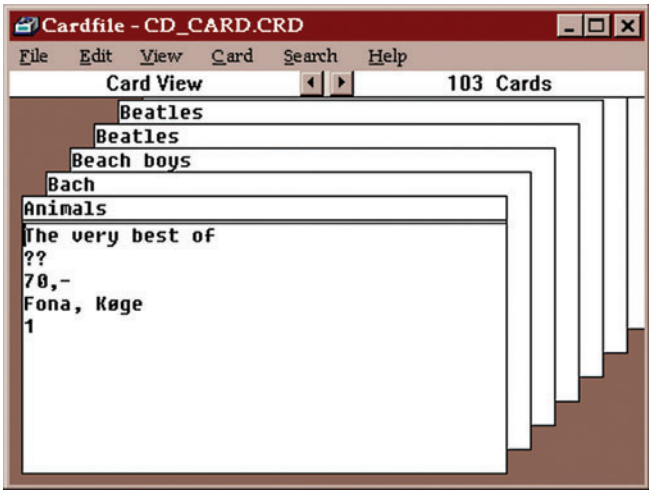


Figure 1 **Remember Cardfile? The original Windows personal information manager looked and acted like paper note cards.**

For example, my Kindle reader reformats the text for the larger type size I prefer, and I read on devices of different sizes (PC, tablet and phone), so page numbers lose their meaning. In a recent presentation, I referred my audience to a Kindle location instead of a page number. And after I introduced them to my classic rock music collection, my daughters now beg me, "Daddy, put on the Beatles' White Playlist."

Kids today grow up with ubiquitous computing, and therefore never learn the connection between digital content and a physical medium. My 2-year-old grandniece and grandnephew, children of my 30-year-old geek nephew, are prime examples. When I gave them music CDs for their first birthday (Raffi singing "Baby Beluga"—revenge on their father for dropping my laptop years ago), their mother said, "I don't think we have a player for those. I'll have to rip them to the Apple music format that we use here." These kids don't have movie DVDs—they're all online, too. They don't have magazines. Nor newspapers. Nor paper photo albums. And now that I think of it, they have darn few paper books, though lots of Kindle editions.

These kids could play music on their dad's iPad before they could walk, though I'm surprised he let the little thugs touch his precious toy. (Perhaps he was hoping they'd break it, so he'd have an excuse to buy this year's improved model.) His daughter, then aged 15 months, got mad and cried when she finger-swiped their big-screen TV and nothing happened.

They consume far more informational content than I did at that age. But that content has been liberated, set free, decoupled from its physical representation. It has become, as Frederick Brooks wrote in "The Mythical Man-Month" (Addison-Wesley Professional, 1995), "nearly pure thought-stuff." The Windows 8 fully digital tenet catches and accelerates this trend.

Toys"R"Us is now selling an Android tablet aimed at kids (with a rubber protective frame) for $150. The cost of producing digital books is tiny compared to the color paper-printing process, although the pricing model hasn't completely caught up yet.

The last generation of humans who will handle paper books (other than as a historical oddity, as we enjoy watching a blacksmith at work) walks the planet today. You tell me: Should we weep, cheer or shrug? ■

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.*

20 WinRT controls at your fingertips

Map control for visualizing geographical summaries or statistcs.

Blazing fast performance – 100,000 points in less than 1 second.

For a limited time, only $99.
syncfusion.com/msdnwinrt

Enterprise
WinRT Components

Syncfusion™