Introducing DXv2

There's no better time to discover DevExpress.

Visual Studio 11 beta is here and DevExpress tools are ready to run.
DXv2 is the next generation of tools that will take your applications to a new level.
Build stunning, touch enabled applications today.
Download your free 30-day trial at **www.DevExpress.com**

DevExpress

THE MICROSOFT JOURNAL FOR DEVELOPERS

APRIL 2012 VOL 27 NO 4

msdn magazine

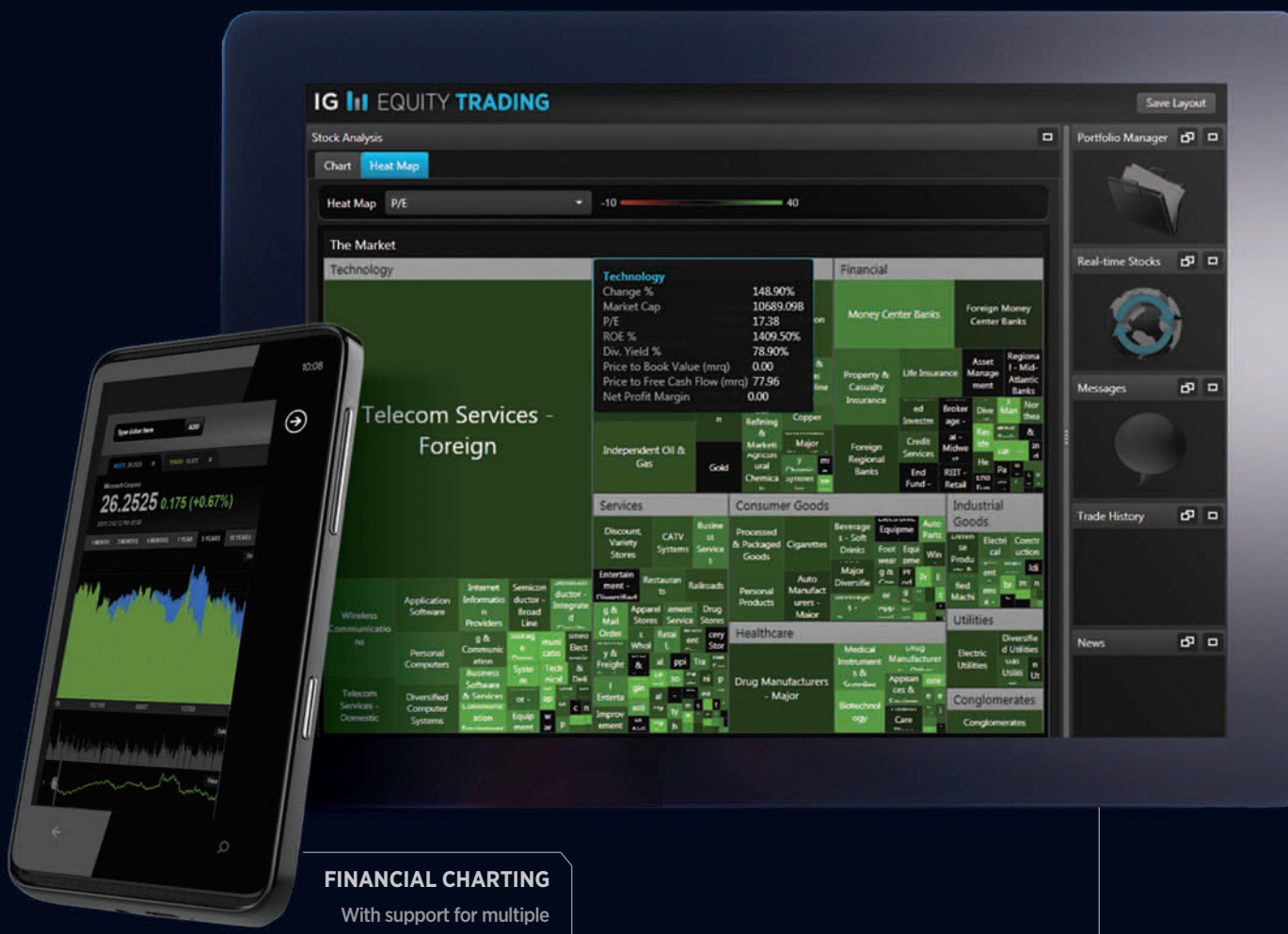
COLUMNS

Microsoft

Deliver the Ultimate User Experience

check out infragistics.com/ultimate

NetAdvantage®
ULTIMATE



FINANCIAL CHARTING

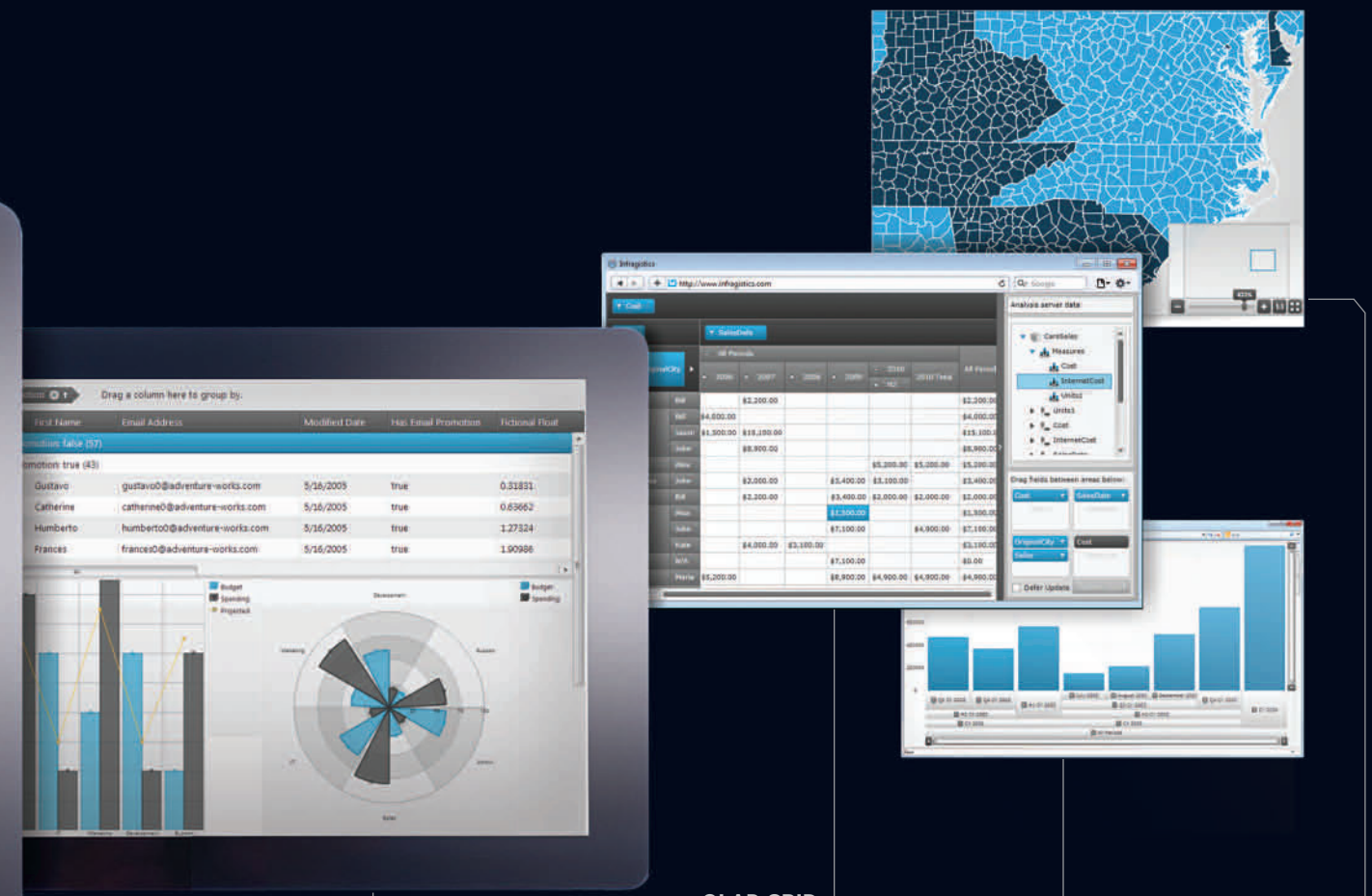
With support for multiple chart styles, and technical indicators built in, financial charting capabilities are on the money.

TREEMAP

Communicate the relative differences in data weight more effectively, with customizable color and flexible layouts.

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



BUSINESS CHARTING

Combine interactive Outlook style grids with rich business charting to deliver a complete portable solution.

OLAP GRID

Provide highly-interactive pivot grid functionality in all of your applications.

OLAP AXIS CHART

Take your data to new depths with the seemingly endless drilldown capability of the OLAP Axis Chart.

GEOGRAPHIC MAP

Plot millions of data point over any type of tile map source or shape file.

msdn® magazine

APRIL 2012 VOLUME 27 NUMBER 4

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

Corporate Address: 1105 Media, Inc., 9201 Oakdale Ave., Ste 101, Chatsworth, CA 91311, www.1105media.com

Media Kits: Direct your Media Kit requests to Matt Morollo, VP Publishing, 508-532-1418 (phone), 508-875-6622 (fax), mmorollo@1105media.com

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

List Rental: This publication's subscriber list, as well as other lists from 1105 Media, Inc., is available for rental. For more information, please contact our list manager, Merit Direct. Phone: 914-368-1000; E-mail: 1105media@meritdirect.com; Web: www.meritdirect.com/1105

All customer service inquiries should be sent to MSDNmag@1105service.com or call 847-763-9560.

Microsoft

BPA WORLDWIDE

Printed in the USA

LEADTOOLS®
MEDICAL SDKs

DICOM SDK / PACS SDK

3D VOLUME RECONSTRUCTION

MEDICAL IMAGE VIEWER CONTROL / MEDICAL WEB VIEWER FRAMEWORK / MEDICAL WORKSTATION

CCOW / PRINT TO PACS

MEDICAL IMAGE PROCESSING / ANNOTATIONS

C++ /.NET / SILVERLIGHT / WINDOWS PHONE
C DLL / WCF SERVICES / WF ACTIVITIES / WPF / ASP.NET & COM

DOWNLOAD OUR 60 DAY EVALUATION
WWW.LEADTOOLS.COM

LEAD TECHNOLOGIES INCORPORATED

SALES@LEADTOOLS.COM
800.637.1840

EDITOR'S NOTE

MICHAEL DESMOND

More of What You Came For

Since 1986, *MSDN Magazine* (and its predecessor *Microsoft Systems Journal*) has been helping to inform, educate and empower professional developers working with Microsoft tools and platforms. Behind the magazine's success has been a single-minded focus on delivering actionable guidance and insight to developers. The hands-on tutorials and how-to articles that appear in each issue of *MSDN Magazine* show not only what you can do with the latest tools and technologies, but how you can get it done.

It's a proven model, and one that's worked very well for us over the years. The biggest frustration, frankly, is addressing the huge range of issues and technologies our readers are engaged with. There's only so much we can do in the pages of content we print and parcel to each of our subscribers every month. Which is why we've been busy extending our coverage beyond the print magazine to include exclusive Web features and columns.

> We're always working to bring new content and greater value to *MSDN Magazine*, and I expect you'll be pleased with the results before we're done.

If you haven't already, check out the *MSDN Magazine* Web site at msdn.microsoft.com/magazine. As ever, you'll be greeted by the features and columns present in the current print issue of the magazine. So you'll see Brandon Satrom's latest HTML5 feature ("Using CSS3 Media Queries to Build a More Responsive Web") at the top of the page, along with a pair of articles from Daniel Moth that explore the powerful Accelerated Massive Parallelism features in the latest versions of C++. Of course, our columnists are also represented, with folks such as Julie Lerman, Dino Esposito and Charles Petzold breaking down a wide variety of technologies.

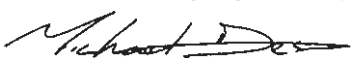
But look around the page and you'll find something new: An online content section that presents additional features and columns published on the *MSDN Magazine* Web site. Each month, you'll find exclusive content, including monthly columns like Rachel Appel's Web Dev Report, which digs into HTML5 development, and Bruno Terkaly's Azure Insider column, which tracks the fast-changing arena of Windows cloud development.

You can also expect to find two or three online-exclusive features at the *MSDN Magazine* Web site each month. In past months we've published features on Agile-based team development, Windows Phone application design, XAML programming techniques and, of course, HTML5. Going forward, we'll use the *MSDN Magazine* Web site to let us further explore topics addressed in print. The ability to follow up our print coverage with Web-exclusive content enables us to address the full depth and context of many of the technical issues presented in our pages.

And believe me when I say, there's more where that came from. We're always working to bring new content and greater value to *MSDN Magazine*, and I expect you'll be pleased with the results before we're done.

We'll continue to improve Web site. If you've visited the *MSDN Magazine* homepage lately, you've probably noticed some subtle changes, like streamlined access to magazine back issues from the Issues link at the top. Or the new By the Numbers department at the bottom of the page, which offers a glimpse into things like what articles your fellow developers are reading and the specific issues or topics they're exploring every month.

The work on the *MSDN Magazine* Web site is ongoing. And we fully welcome your advice and input as we expand the depth and breadth of coverage for our readership of expert developers. What would you like to see changed or added on the Web? What topics would you like to see addressed in our ongoing Web-only coverage? Drop me a line at mmeditor@microsoft.com.



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

axosoft
800.653.0024

Everything You Need to Ship Software OnTime

Powerful bug tracking.

Manage bugs, defects, & issues with automatic notifications & alerts. Customize the tracker to match your development workflow. Track features and requirements, too—with drag and drop subitems support.

The **best** Scrum tool.

Utilize Scrum development or another agile process? OnTime is easy to adapt, and includes intelligent burndown & burnup charts, sprint & release backlogs, and the best interactive planning board out there.

A **beautiful & fast** UI.

You shouldn't spend all of your time in a software dev tool when you could be actually coding. OnTime's UI is fast and simple, and you can get what you need done right away. We think you'll really like it.

Plus: OnTime Mobile • GitHub Integration • Customer Portal & Help Desk • Built-in Scratchpad Remote Server • Windows, Visual Studio & Eclipse Clients • Premium Hosting • SDK • and more!

Visit axosoft.com to learn more.

Free 2-user hosted license. **Forever.**

The best way to find out how OnTime can help your team is to sign up for a free 2-user license. Hosted on our reliable, secure OnTime Now! service, you can get started right away. Plus, for 30 days you can add up to 10 total users—but you'll still have 2 users forever.

Did you **know?**

OnTime is built on the Microsoft .NET framework, and it has a Microsoft SQL Server back-end. You can use OnTime as a hosted service or install it in your own environment. With OnTime's APIs your dev team can extend the functionality of OnTime inside your apps.

Find us on:

@axosoft Axosoft OnTime /axosoft

axosoft OnTime11

Team up. Collaborate. Build. **Ship great software OnTime.**

DINO ESPOSITO

Long Polling and SignalR

We are building more and more sophisticated Web applications on top of a protocol that, paradoxically, was conceived and designed for much simpler forms of interaction. HTTP has no built-in support for state or even security. Its basic assumption is that the client places a request and the Web server issues a response. In sum, that means no request, no response.

Given the current penetration of the Web and the ubiquity of Web solutions, changing the pillars of the Web (HTTP, HTML, JavaScript) is out of the question. But should we consider improving some of those pillars? Of course. Modern applications have specific requirements that push Web protocols and languages to their limit—or beyond.

> There's always a client request at the core of any solution based on polling.

In recent installments of this column, I discussed a handcrafted implementation of a mechanism that polls the server and reports changes to the client. Last month, I implemented the same idea using the services of an emerging library—SignalR. Now I'll provide a brief overview of the techniques and a summary of the options you have today. I'll put SignalR in the spotlight and look at its implementation—and some of its magic.

Polling Considerations

Given the HTTP request/response constraint, polling is the only possible way to set up live communication between a client and Web server. The client pushes requests at its convenience and the server diligently replies. The key to evaluating the effectiveness of polling is the actual meaning you assign to the expression "at its convenience." There's always a client request at the core of any solution based on polling. The request stands until the server has replied. Any pending request consumes a browser connection and, more importantly, engages a server thread, making that valuable resource unavailable to other requests. In a nutshell, too-frequent requests build pressure on the Web server.

But isn't the server's ability to handle a growing number of requests the whole idea of scalability? In fact, polling doesn't create new scalability concerns, but it does add a substantial number of new requests to the server that must be taken into account to ensure a scalable and high-performing server. Let's see how we might implement polling.

AJAX Polling

In my December 2011 (msdn.microsoft.com/magazine/hh580729) and January 2012 (msdn.microsoft.com/magazine/hh708746) columns, I presented a framework for controlling the progress of a remote operation. The entire framework was based on AJAX calls. First, the client invokes a server endpoint to start a potentially lengthy task; next, it sets up a timer to place concurrent calls to another endpoint every 500 milliseconds. As the server task does its job, it updates a known location. The server endpoint that's invoked periodically simply checks for data in that location and returns any content it finds to the client.

This AJAX-polling pattern works beautifully and is widely employed in many industry scenarios. Most sites that provide news updates or live scores follow this pattern. In my articles, I simply adapted the pattern to the specific scenario of monitoring the progress of a remote operation.

In the end, the problem with AJAX polling is figuring out the right refresh interval—one that represents a good balance between pressure on the server and accuracy of information reported to users. An effective AJAX-polling solution gives a concrete meaning to "at the client's convenience" in sending requests.

Figure 1 **A SignalR Class that Performs a Multistep Operation**

```
public class BookingHub : Hub
{
  public void BookFlight(String from, String to)
  {
    // Book first leg
    Clients.displayMessage(
      String.Format("Booking flight: {0}-{1} ...", from, to));
    BookFlightInternal(from, to);

    // Book return
    Clients.displayMessage(
      String.Format("Booking flight: {0}-{1} ...", to, from));
    BookFlightInternal(to, from);

    // Book return
    Clients.displayMessage("Charging credit card ...");
    ChargeCreditCardInternal();

    // Some return value
    Clients.displayMessage("Flight booked successfully.");
  }
}
```

Bring your XML development projects to light with the complete set of tools from Altova®

ALTOVA® missionkit®

Experience how the Altova MissionKit®, the integrated suite of XML, database, and data integration tools, can simplify even the most advanced XML development projects.

NEW in Version 2012:

- HTML5 & CSS3 support
- EPUB 2.x support
- Streaming reading/writing of files for large-scale ETL apps
- JDBC support (in addition to ADO and ODBC)
- New, native Java APIs

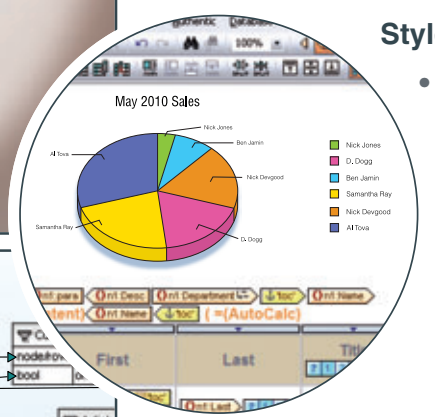
The Altova MissionKit includes multiple intelligent XML tools:

XMLSpy® – industry-leading XML editor

- Support for all XML-based technologies
- Editing of HTML4, HTML5, XHTML, CSS
- Graphical editing views, powerful debuggers, code generation, & more

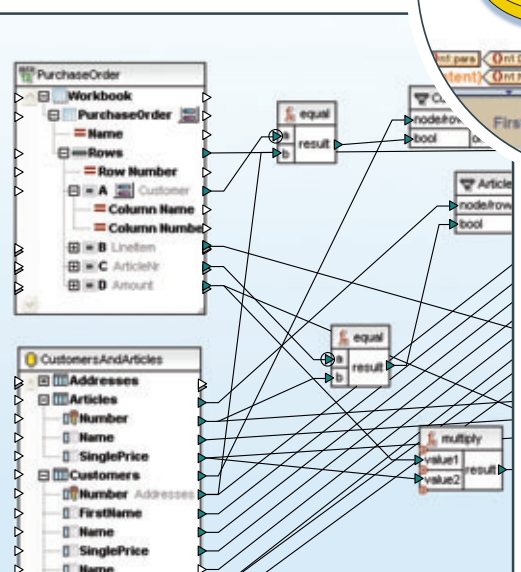
MapForce® – graphical data mapping & ETL tool

- Drag-and-drop data conversion with code generation
- Mapping of XML, DBs, EDI, Excel®, XBRL, flat files & Web services



StyleVision® – visual stylesheet & report designer

- Graphical stylesheet and report design for XML, XBRL & databases
  - Report designer with chart creation
  - Output to HTML, PDF, Word & eForms



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 MissionKit XML tools

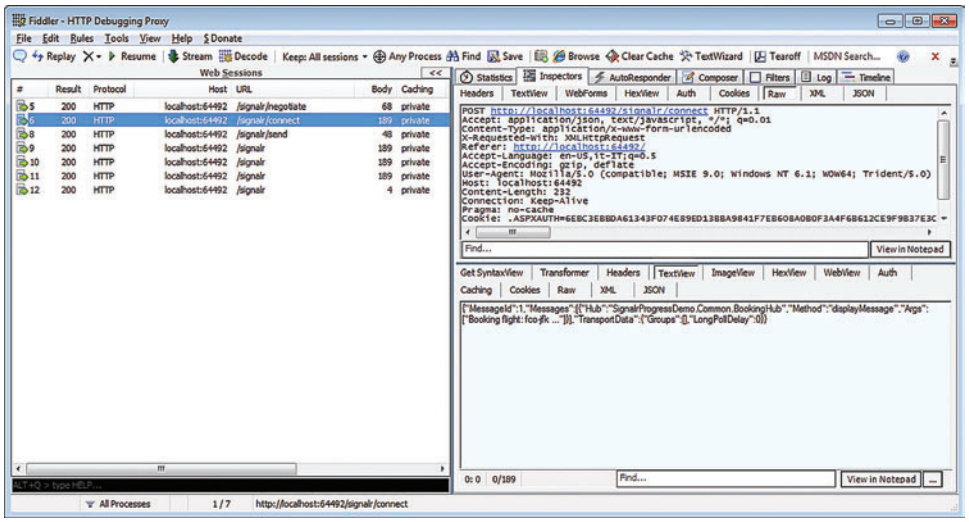


Figure 2 **The Full Stack of HTTP Requests for the Flight-Booking Operation**

Long Polling

Before AJAX, developers used the meta refresh HTML tag to instruct browsers to refresh a page every given number of seconds. With AJAX polling, you achieve the same thing, but in a much smoother way.

For a growing number of applications, however, this form of polling isn't enough. AJAX polling almost inevitably introduces a delay between the occurrence of an event on the server and the notification of the event to the client. By tuning the frequency of updates, you can work out good solutions, yet AJAX polling can't deliver that constant and continuous connection that many (mostly social) applications seem to need today.

The solution these days seems to be long polling. Curiously, when AJAX emerged years ago, smarter AJAX polling replaced long polling because the latter was not very effective over the Web. Long polling is about doing over the Web the same things you do in a desktop scenario. With long polling, the client places the request and the server doesn't reply until it has information to return. The Web client keeps a pending connection that's closed only when some valid response can be returned. That's exactly what you want nowadays—except that it has the potential of slowing down the Web server.

Long polling places a smaller number of requests to the server compared with AJAX polling, but each request could take much longer. Over time, this can be detrimental to the health of the Web server because it keeps server threads engaged until a response can be generated. With fewer worker threads available at a given time, the Web server inevitably gets slower in responding to any other requests it receives. To be effective, long polling needs some serious implementation work and advanced multithreaded and parallel programming skills. So, for years, long polling wasn't really an option, but it didn't matter because there was no demand for continuous connectivity.

Today, with the Task Parallel Library in the Microsoft .NET Framework 4 and other facilities in ASP.NET for building asynchronous HTTP handlers, long polling has become a viable option. Overall, handcrafting a long-polling framework is still harder than a corresponding AJAX-polling framework. You need a well-designed server environment that doesn't tax the Web server and a client environment that can support long polling. Long polling, in fact, refers to a macro client/server operation that develops over the Web and necessarily over a number of classic HTTP request/response packets. The client must be smart enough to reissue an immediate new request until the macro operation terminates. I'll return to this point later.

SignalR to the Rescue

SignalR is a Microsoft framework specifically designed to facilitate real-time client/server communication. It provides an effective implementation of long polling that doesn't have a deep impact on the server, and at the same time ensures that clients are appropriately updated about what's going on remotely. **Figure 1** shows an excerpt from the code I presented in my last column. The BookingHub class is the method you invoke from the Web browser to start the macro operation "book the flight."

This is clearly a multistep operation, and for each step the class sends a notification to the client. How many HTTP requests are involved? Let's take a look with Fiddler (see **Figure 2**).

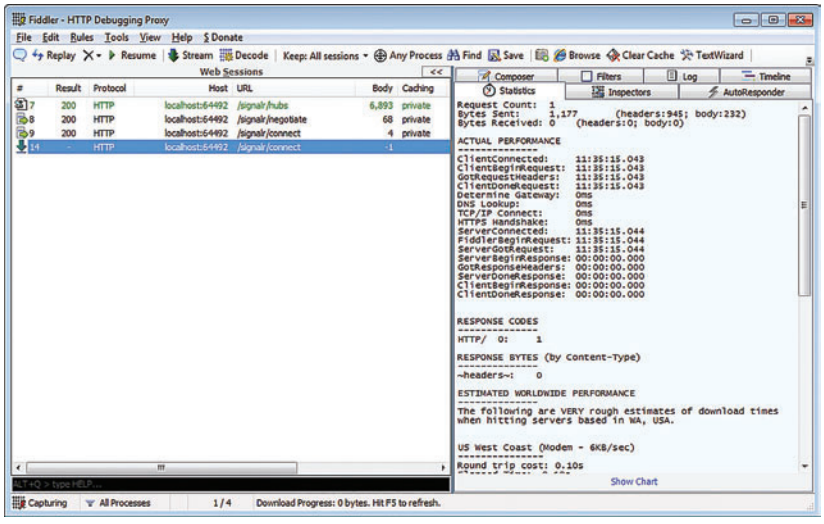


Figure 3 **Reiterating Connection Requests**

Inside a SignalR Operation

The first request is triggered when you call the method start from the client page; this identifies the client to the SignalR back end. It should be noted that the first request is a special negotiation request. It will always be AJAX regardless of the

Powerful Tools for Developers

v4.5!

High-Performance PDF Printer Driver

- Create accurate PDF documents in a fraction of the time needed with other tools
- WHQL tested for all Windows 32 and 64-bit platforms
- Produce fully compliant PDF/A documents
- Standard PDF features included with a number of unique features
- Interface with any .NET or ActiveX programming language

v4.5!

PDF Editor for .NET, now Webform Enabled

- Edit, process and print PDF 1.7 documents programmatically
- Fast and lightweight 32 and 64-bit managed code assemblies for Windows, WPF and Web applications
- Support for dynamic objects such as edit-fields and sticky-notes
- Save image files directly to PDF, with optional OCR
- Multiple image compression formats such as PNG, JBIG2 and TIFF

New!

PDF Integration into Silverlight Applications

- Server-side PDF component based on the robust Amyuni PDF Creator ActiveX or .NET components
- Client-side C# Silverlight 3 control provided with source-code
- Optimization of PDF documents prior to converting them into XAML
- Conversion of PDF edit-boxes into Silverlight TextBox objects
- Support for other document formats such as TIFF and XPS

OCR Module available with royalty-free licensing!

The new OCR Module from Amyuni enables developers to:

- Convert non-searchable PDF files into searchable PDFs
- Create searchable PDF documents out of various image formats such as multi-page TIFF, JPEG or PNG while applying text recognition
- Compress image based PDF documents using high compression JBIG2 or more standard CCITT, JPEG and PNG compression formats

The Amyuni OCR module is based on the Tesseract Library with the Amyuni PDF technology being used to process and create the PDF documents.

Learn more at www.amyuni.com

Windows 7

Certified for Windows Server 2008

More Development Tools Available at:

www.amyuni.com

USA and Canada
Toll Free: 1 866 926 9864
Support: (514) 868 9227
Info: sales@amyuni.com

Europe
Sales: (+33) 1 30 61 07 97
Support: (+33) 1 30 61 07 98
Customizations: management@amyuni.com

AMYUNI Technologies

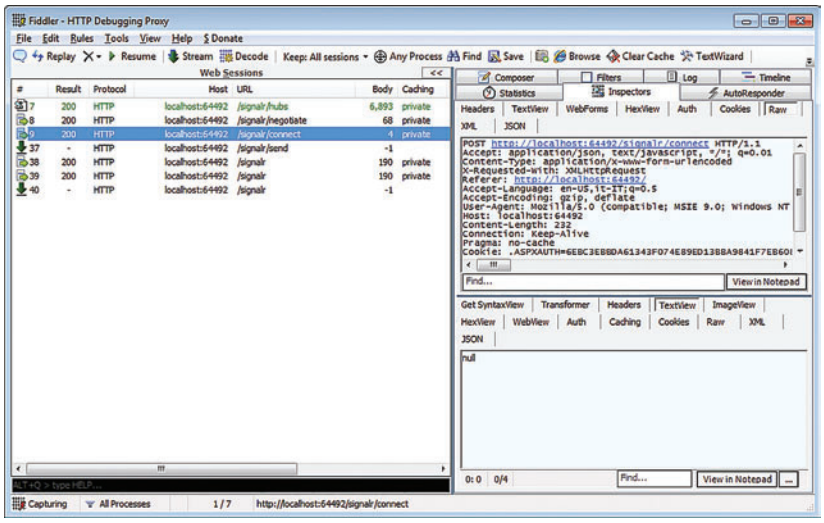


Figure 4 **A Multistep Operation Is Taking Place**

final transport negotiated for the connection. Any SignalR Web page includes code like the following that runs when a page loads:

```
$(function () {
  var bookingHub = $.connection.bookingHub;
  bookingHub.start();
}
```

In this way, the page declares its intention to open a connection to invoke the services of the server-side bookingHub object. Note that it's SignalR that does the magic of creating a client object according to the public interface of the server-side hub object. The name of the JavaScript object matches the name of the server object. However, you can use the HubName attribute to modify the name being used on the client:

```
[HubName("booking")]
public class BookingHub : Hub
{
  ...
}
```

The startup request returns a client ID that the client will pass along with any successive requests.

```
{"Url":"/signalr","connectionId":"2a119962-edf7-4a97-954b-e74f2f1d27a9"}
```

Next, the client library places another request for connection. This is the "connect" request, which will result in the OnConnect event being raised for the connection on the server. **Figure 3** shows a couple of connect requests.

The first completed in two minutes; the second is still pending. This is the essence of long polling—the client has a channel continuously open with the server. The server times out the request after two minutes if no action is performed on the server that requires sending data back to the client. In the flight-booking application UI, there's a button the user can click to start booking the flight. When the user clicks the button the following code executes:

```
bookingHub.bookFlight("fco", "jfk");
```

If you created the example application from last month's column, you should have linked this code to the click handler of the page button.

The bookingHub object belongs to a script that SignalR downloads via the signalr/hubs URL, as shown in **Figure 3**. bookingHub is a plain proxy object that hides the complexity of implementing a long-polling pattern. Clicking the button triggers a new request to the server where the client ID is embedded. When the server receives a call from a client that has a pending call, it ends the pending call and starts processing the new request, as shown in **Figure 4**.

Figure 4 shows a pending request (signalr/send), two completed requests and another pending request. The signalr/send request is the original call to the macro operation for booking the flight, whose source code is shown in **Figure 1**.

The code in **Figure 1** calls the client at various stages to notify it of any progress, like so:

```
Clients.displayMessage(...);
```

At the same time a request for signalr/send is placed, another request starts to wait for notifications. This request waits until there's some data to send back. Any call to Clients in the server code completes the request for notification and immediately triggers another one from the client, so the sequence in **Figure 4** means that two steps have been completed and two progress messages have been received by the client. The server process is now busy trying to accomplish the third step. By the end of the code in **Figure 1**, all of the requests in **Figure 4** will have completed and the situation will return to one similar to what's shown in **Figure 2**.

Beyond Long Polling

If you compare the behavior of long polling as implemented in SignalR with the steps I indicated in my articles about AJAX polling, you should recognize a common pattern—the Progress Indicator pattern. AJAX polling and long polling are two different implementations of the same pattern. Which one is more effective depends on the configuration of the Web server and requirements of the application. It's your call. In any case, if you opt for long polling, you need SignalR or an analogous library. If you want to build your own framework, I suggest you opt for the numerous-but-quick calls of AJAX polling versus the fewer-but-longer calls of long polling.

Building an effective long-polling framework can be tricky. With AJAX polling you probably don't get continuous connectivity, but you also don't risk slowing down the server. With this in mind, I probably won't go around and update any real-time Web server I take care of to SignalR. However, once SignalR is released, I don't see any reason not to use it for new applications.

Finally, it's worth mentioning that SignalR now supports other higher-level transports besides long polling. In the latest source, you also find support for WebSockets on Windows 8 servers; Server Sent Events on Chrome, Firefox, Opera and Safari; and Forever Frame on Internet Explorer. The library starts at the beginning of the list and keeps falling back until a supported transport is found. So, in the end, long polling will actually rarely be used in most cases. ■

Dino Esposito *is the author of "Programming ASP.NET 4" (Microsoft Press, 2011) and "Programming ASP.NET MVC 3" (Microsoft Press, 2010), 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: *Damian Edwards*

NEW!

CLOUD SERVER CONTROL AT YOUR FINGERTIPS

Only pay for what you need.
Change your server specifications anytime!

- Adaptable with up to 6 CPU, 24 GB of RAM, and 800 GB hard drive space
- On-the-fly resource allocation – hourly billing
- Dedicated resources with full root access
- Linux or Windows® operating systems available with Parallels® Plesk Panel 10.4
- Free SSL Certificate included
- 2,000 GB Traffic
- 24/7 Hotline and Support
- 1&1 servers are housed in high-tech data centers owned and operated by 1&1

1&1 DYNAMIC CLOUD SERVER

3 MONTHS FREE!*

Base Configuration, then $49/month

NEW: Monitor and manage servers through 1&1 mobile apps for Android™ and iPhone®.

MEMBER OF united internet

1-877-461-2631 www.1and1.com

1-855-221-2631 www.1and1.ca

1&1®

* 3 months free based on the basic configuration ($49/month) for a total savings of $147. Setup fee and other terms and conditions may apply. Visit www.1and1.com for full promotional offer details. Program and pricing specifi cations and availability subject to change without notice. 1&1 and the 1&1 logo are trademarks of 1&1 Internet, all other trademarks are the property of their respective owners.

Develop Once, Deploy Anywhere.

ComponentArt's unique technology targets the widest range of devices with a single code base. Develop once in Visual Studio and XAML, deploy anywhere in a variety of formats.

HTML5

HTML5 is supported by virtually any device: Apple iPad & iPhone, Android Tablets & Phones, Windows Tablets & Phones, Blackberry Playbook & Phones, and any PC or Mac.

XAML/WinRT

Windows 8 features native XAML support and a new WinRT library. Deploy immersive Metro XAML apps through ComponentArt Data Visualization for XAML/WinRT.

Silverlight & WPF

Classic Windows interface continues to be supported through mature .NET development tools: ComponentArt Data Visualization for Silverlight & WPF.

Single code base. Any output format.

Powered By: Microsoft Visual Studio · ComponentArt Data Visualization · ComponentArt Dashboard Server

Sounds unbelievable?

See for yourself at www.ComponentArt.com

JULIE LERMAN

Let Power Users Create Their Own OData Feeds

I have a client for whom I develop many in-house business applications. The company also has a resident geek, a clever young guy who leverages his Web 2.0 background to create some in-house solutions as well. I code for them using Microsoft .NET Framework tools against a SQL Server 2008 database. He develops PHP apps that use an online MySQL database. A few years ago he showed me what he was working on and I noticed dropdown lists containing data that duplicated what my apps create and consume in SQL Server. WCF Data Services was still in its early stages (known then as "Astoria"), but I saw a great opportunity to try this technology out as a bridge between our development projects.

I proposed creating a read-only service from the SQL Server data that he could consume in his applications, eliminating the need to maintain the duplicate data in the MySQL database. He loved the idea. A few days later, Microsoft released its first PHP toolkit for consuming Astoria feeds. This developer was one of the first toolkit users and we had our collaborative solution up and running very quickly.

> Data Explorer allows users to access a variety of data sources and build views from them using a relatively simple interface.

Fast forward … Astoria is now WCF Data Services, its output has been encapsulated in a specification called OData (odata.org) and there's an OData SDK for PHP (bit.ly/xW8sJf). My client's service has evolved after numerous requests that I expose more data from the SQL Server database for his applications. But there's a problem. Each time my client needs something new added to the service, I have to create an appropriate view in the database, open Visual Studio, modify the data model to include an entity that maps to this view, then compile and redeploy this new service over the VPN to his Web server. The process is fairly time-consuming and can be a bit of a drag—especially if I'm getting close to a Data Points column deadline!

This article discusses Microsoft Codename "Data Explorer." All information is subject to change.

Help Arrives: SQL Azure Labs Codename "Data Explorer"

At the SQLPass Summit in October 2011, Microsoft announced a new tool under development that's currently called Microsoft Codename "Data Explorer." Here, I'll simply call it Data Explorer.

Data Explorer allows users to access a variety of data sources (SQL Azure, SQL Server, HTML from Web pages, OData feeds, text files and Excel files, to name a few) and build views from them using a relatively simple interface. The UI lets you slice, dice, filter, rename and combine data. You can mash up data from disparate file types to, for example, create a view that joins your SQL Server data to statistics listed on a Web page, and then present the results as an OData feed, an Excel file or any of a number of other formats.

You can find lots of walk-throughs and videos of Data Explorer features on the team's blog (blogs.msdn.com/dataexplorer) or at the SQL Azure Labs (sqlazurelabs.com). There's a Web-based client (referred to as a cloud service) and a desktop client. The cloud service currently lives in the SQL Azure Labs as a service and you must sign up for access (at bit.ly/pl8Oug). You can download the current version of the desktop app from bit.ly/shltWn. As of this writing, these tools are still in early development stages, and I anticipate many improvements and benefits to come.

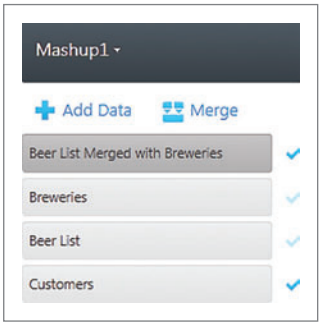


Figure 1 **Four Data Sets that Have Been Created in Mashup1**

One of the Data Explorer features rang out as a way to simplify my process of adding more data to the Data Service I maintain for this client—the ability to publish the results of the slicing and dicing as an OData feed. Today (early January 2012, as I write this column) publishing is supported by the cloud service but not yet by the desktop client. While my eventual goal is to do all of this work on my client's internal network using its SQL Server database and Web

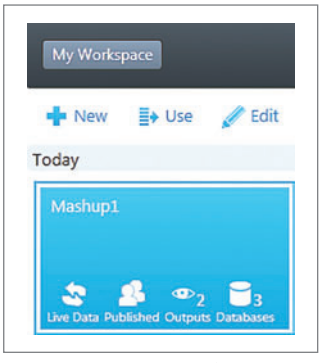
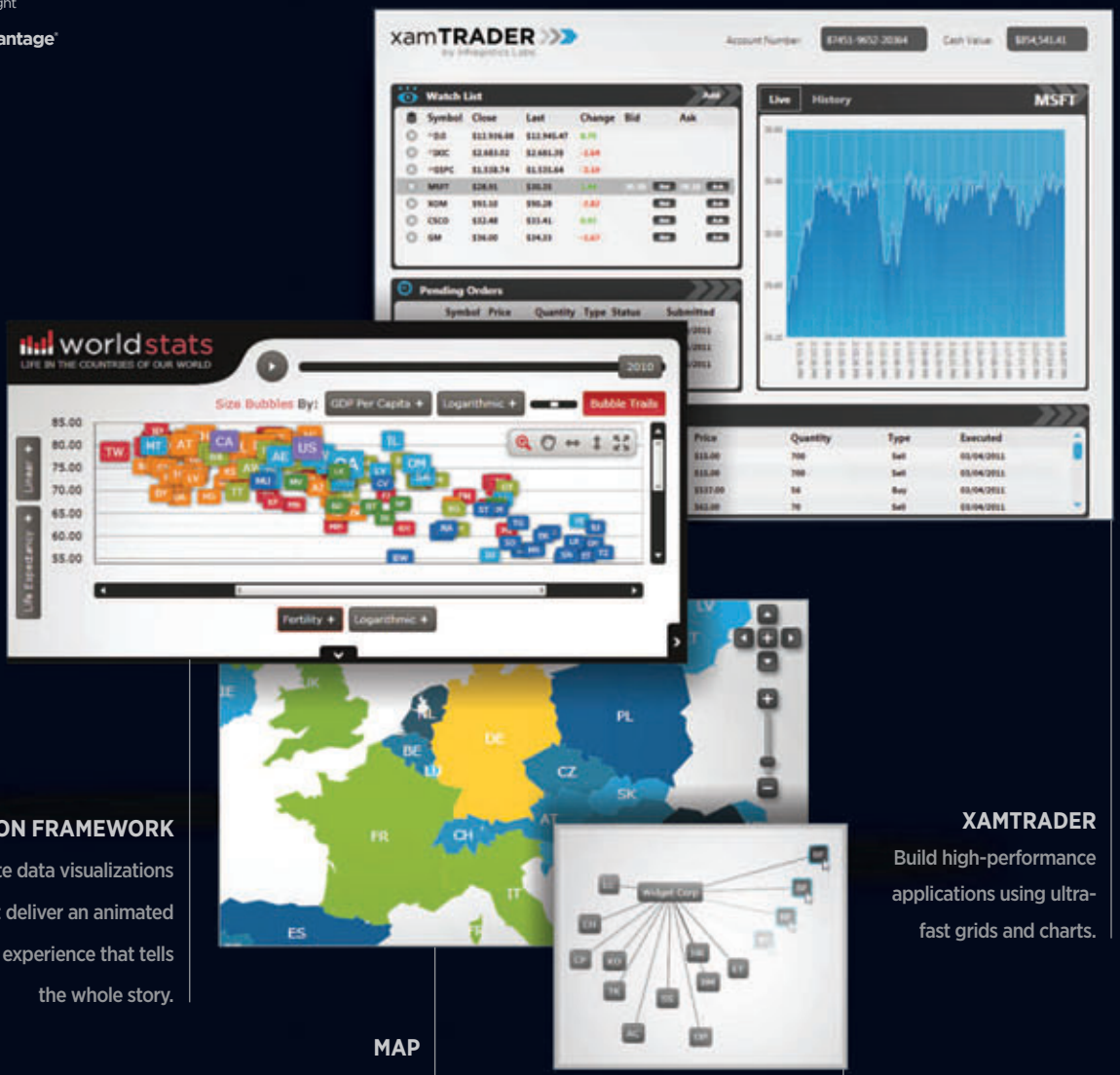


Figure 2 **Overview of Mashup1 in My Workspace**

XAML-IFY YOUR APPS

check out infragistics.com/xaml

- **NetAdvantage®** for Silverlight Data Visualization
- **NetAdvantage®** for WPF Data Visualization
- **NetAdvantage®** for Silverlight
- **NetAdvantage®** for WPF



MOTION FRAMEWORK

Create data visualizations that deliver an animated user experience that tells the whole story.

MAP

Ensure your geospatial data really goes places with a feature-laden, interactive Map Control for your applications.

NETWORK NODE

Help your users make the connection with visual representations of simple or complex network relationships.

XAMTRADER

Build high-performance applications using ultra-fast grids and charts.

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 **Two Data Sets Exposed as Collections in My Published Mashup**

```
<?xml version="1.0" encoding="UTF-8"?>
<service xmlns="http://www.w3.org/2007/app" xmlns:atom="http://www.w3.org/2005/
Atom" xml:base="https://ws39047873.dataexplorer.sqlazurelabs.com/Published/
CustomersAndBeerData/Feed/">
  <workspace>
    <atom:title type="text">Default</atom:title>
    <collection href="Beer List Merged with Breweries">
      <atom:title type="text">Beer List Merged with Breweries</atom:title>
    </collection>
    <collection href="Customers">
      <atom:title type="text">Customers</atom:title>
    </collection>
  </workspace>
</service>
```

service, I've explored the possibility of using the cloud service and a SQL Azure database. This is what I'll share with you. I won't bother with a step-by-step walk-through, as that's readily available in the resources I mentioned earlier.

Your workspace can contain multiple mashups and each one can contain a number of resources. **Figure 1** shows four resources I created in a mashup called Mashup1. Three of the resources expose a view of Customers, a list of Beer data and a list of Breweries, while the final resource (Beer List Merged with Breweries) exposes a merged and flattened list of the breweries along with the beer each produces. I had to create the Beer List and Breweries resources in order to merge them into yet another resource. The state of the "check mark" icon to the right indicates what will and won't be exposed in the published data. The dimmed check marks next to Breweries and Beer List indicate that these will not be exposed. As you can see in **Figure 1**, I plan to publish the Customers and the Beer List Merged with Breweries, but not the building blocks I used to create the merged set.

If you look at the mashup's overview in my Workspace, as shown in **Figure 2**, you can see that Mashup1 has two outputs, representing the two resources I've chosen to expose.

When I publish Mashup1 using the Data Explorer tools and then browse the resulting OData feed, you can see that the service does indeed expose those two resources and nothing more (see **Figure 3**).

The Data Explorer UI enables you to build up complex queries visually, but because I'll be building this data feed, it's easier for me to just create a view in the database. So my plan is that each time my power user wants access to more data in his service, I'll create a database view, add it as a resource in my mashup and republish the feed.

In this case, because I want my OData feed to just reflect the changes in the schema on demand, I can simply do that. But you need to be careful because apps that are relying on the feed might break if the feed introduces non-additive or certain other changes.

As another approach, however, Data Explorer also lets me shield my API/feed surface from any changes in the schema. I simply have to update the mashup to account for the schema changes, while keeping the shape of my outputs unchanged.

Security-Minded Layers

I built these resources directly from my SQL Azure database. My power user could do the same, but I don't want him to directly access the database. Instead, I'll let him have his own Workspace and build his own mashup from the one I published.

Figure 4 shows all the types of data you can use to create resources in a mashup. SQL Databases can be either a SQL Server database or a SQL Azure database.

> Your workspace can contain multiple mashups and each one can contain a number of resources.

Among the possible data sources are an OData Feed and an existing mashup. These make it possible for the power user to build a mashup from my published feed.

He can add a dataset based on my OData feed to a mashup in his own workspace and then massage that view for his own needs. Data Explorer lets him hide columns he's not interested in, build concatenated fields, change how his service displays field names and so much more. If, for example, he initially thinks he doesn't

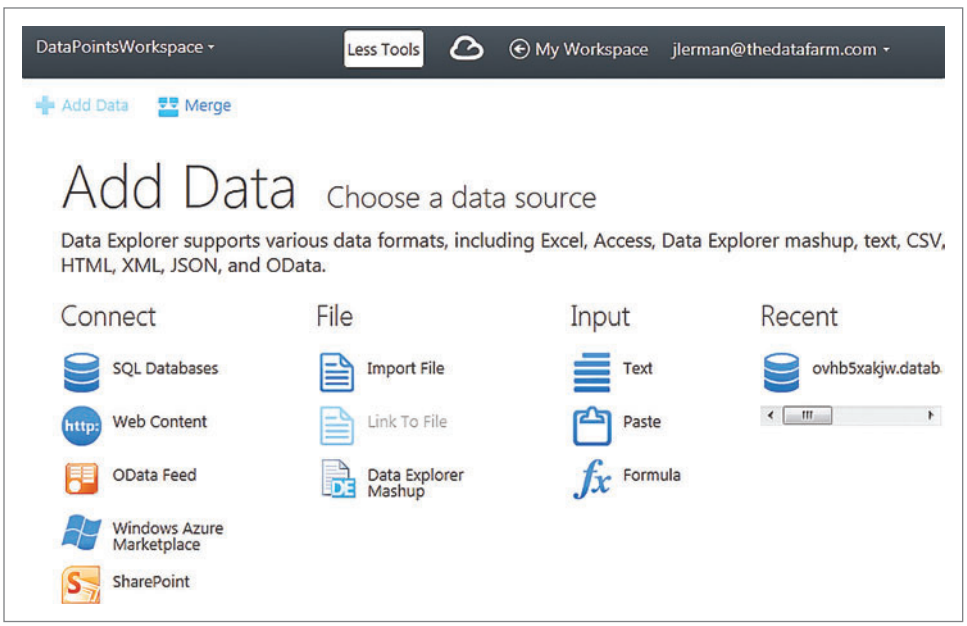


Figure 4 **Options for Adding Data to a Mashup**

Modern Applications Need Sophisticated Controls

check out infragistics.com/windowsforms

NetAdvantage® for Windows Forms

SMART, STYLISH UI SOLUTIONS

The most sophisticated set of continually enhanced, performance-dominating Windows Forms UI controls available anywhere.

GANTT VIEW

Plan for success Microsoft Project-style, with Timeline and Grid Views, Styling capabilities and support for Custom Columns.

CHARTING

With advanced graphics and more than 50 2D and 3D Chart Types, easily create and customize the best visualizations of your information.

GRID

Be more productive with this hierarchical data grid control as the backbone of your applications.

Want to ensure your WPF and Windows Forms application UIs are fully optimized?

TURN THE PAGE!

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

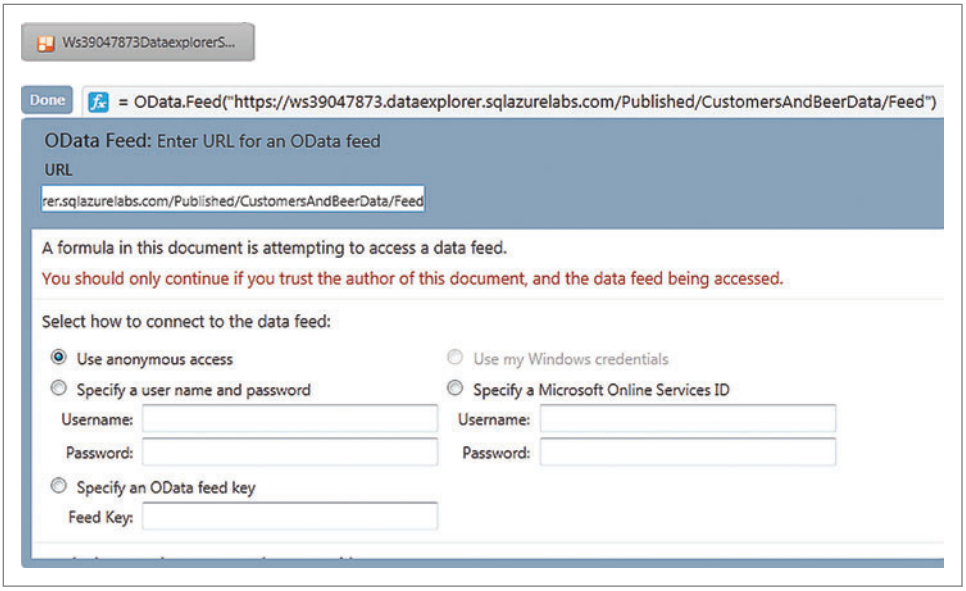


Figure 5 **Adding an OData Feed to a Mashup**

need a column and later decides he does want access to it, he can simply modify his own mashup without worrying about whether I'm abroad at a conference or out on a bike ride.

The database stays secure, but what about the data feed I'm creating? Data Explorer already has a number of security features built in, and Microsoft is working on enriching security even more. One security path is to grant permissions to a user with an existing Data Explorer account to access your workspace as an Owner, an Author or a Consumer in the workspace. Were I to grant my power user Author permissions, in his own workspace he could then create and publish his own mashups based on my feed.

I could also provide the power user with a Feed Key. Each workspace has a single Feed Key that lets anyone access any feed published from a workspace. When the power user accesses my OData feed to build his own mashup, he'll be required to supply whatever credentials I've specified. **Figure 5** shows the credentials being requested when the user tries to add my CustomersAndBeerData feed to his own.

Once he's pulled my feed into his own mashup, he can reshape the output of the feed, even combining it with other data sources if he wishes. When he's satisfied, he can publish his results to an OData feed of his own and consume it from his application using credentials that he specifies in his own workspace.

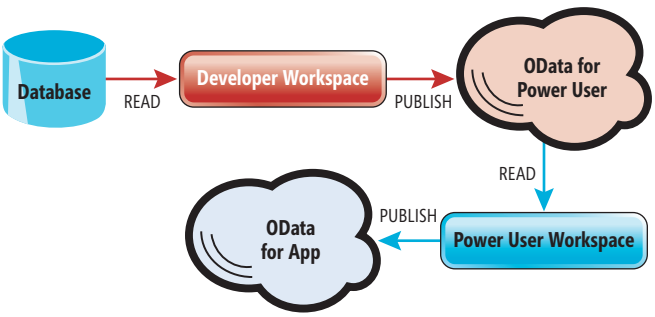


Figure 6 **Workflow from Database to Application**

Data Explorer

By Miguel Llopis and Faisal Mohamood

Microsoft Codename "Data Explorer" is a new tool that enables self-service Extract, Transform and Publish (ETP). Data Explorer simplifies the creation of data mashups from a combination of different data sources, such as Excel, Access, SQL Server, the Web (HTML, Web APIs, OData) and more. The main goal for Data Explorer is to make the acquisition, transformation and composition of data easy and seamless in order to publish the data into end-user tools such as Excel/PowerPivot or to power custom applications.

Data Explorer is currently available as a lab at SQL Azure Labs and supports both on-premises as well as cloud data sources. Visit dataexplorer.sqlazurelabs.com to learn more and try out this innovative Windows Azure service.

Data Explorer is not yet a released product and will continue to evolve.

Worth the Wait

In the end, when the power user wants additional OData for his application, my involvement is greatly reduced. I still need to add a view in the database, then add a new resource to my mashup online, then republish. But this workflow, visualized in **Figure 6**, is much more appealing to me than having to open the project in Visual Studio, update the data model, rebuild, log in to the VPN and, finally, push the new assembly to my client's server.

Data Explorer is not yet a released product and will continue to evolve. But having worked through this scenario using the currently available preview, I'm truly looking forward to shifting to this solution when Data Explorer is officially released. And while I'm currently focused on a very narrow use case, I'll be keeping an eye on blogs.msdn.com/dataexplorer for updates. ■

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" (O'Reilly Media, 2010) and "Programming Entity Framework: Code First" (O'Reilly Media, 2011). Follow her on Twitter at twitter.com/julielerman.*

Thanks to the following technical experts for reviewing this article: *Miguel Llopis and Faisal Mohamood*

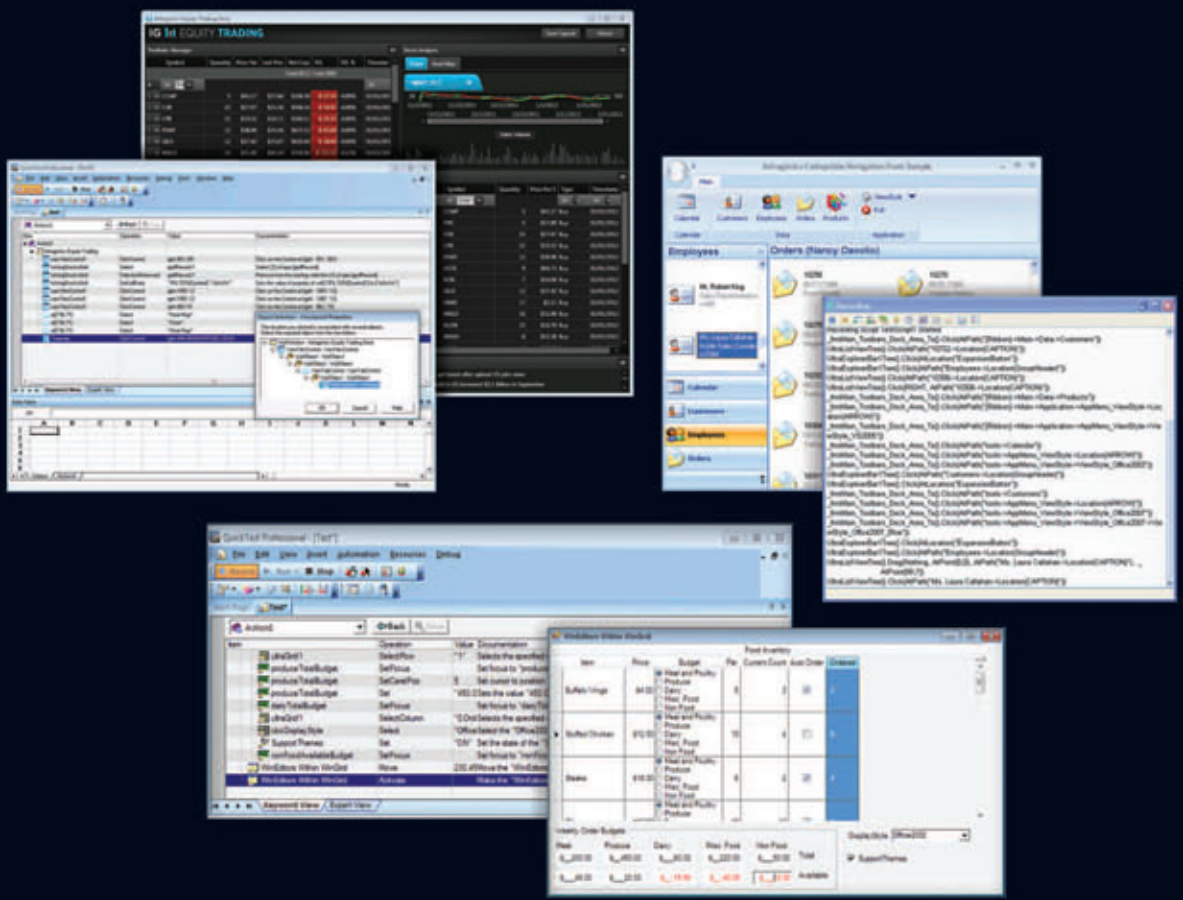
Application Assurance

check out infragistics.com/testadvantage

TestAdvantage®
for Windows Forms
for HP QuickTest Professional software

TestAdvantage®
for Windows Forms
for IBM Rational Functional Tester

TestAdvantage®
for WPF
for HP QuickTest Professional software



ARE YOUR WPF AND WINDOWS FORMS APPLICATIONS FULLY OPTIMIZED?

Reduce the risks associated with project development and deployment of your applications with TestAdvantage for WPF and Windows Forms, the first out-of-the-box tools to enable automated of testing of applications built with NetAdvantage for WPF and Windows Forms toolsets. TestAdvantage for WPF (for HP QuickTest Professional) and Windows Forms (for HP QuickTest Professional or IBM Rational Functional Tester) extend the capabilities of your testing environment to ensure your applications are designed and optimized for success.

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

Using CSS3 Media Queries to Build a More Responsive Web

Brandon Satrom

A few years ago, I was approached by a client who wished to build a mobile-only Web site for its young, tech-savvy user base. The project was actually pitched to us as an "iPhone site," and came with a fully conceived design. Excited to create such a cutting-edge mobile app, we began in earnest and made decent headway—until it came time to consider other mobile devices on the market. This client wanted support for just about all of them. Thus began our valiant effort to answer two of the most important questions Web developers face when attempting to build mobile-tailored experiences: How do you know when to change the experience for a given mobile browser, and how do you go about making that change when one is required?

This article discusses Internet Explorer 10 Platform Preview. All information is subject to change.

This article discusses:

- Responsive Web design
- Getting started with media queries
- Modifying styles for mobile devices
- Developing for mobile first
- Working with media query listeners

Technologies discussed:

CSS3, Internet Explorer 9, Internet Explorer 10 Platform Preview, JavaScript

Code download available at:

code.msdn.microsoft.com/mag201204HTML5

Responsive Web Design

These questions are at the heart of responsive Web design, a term coined by Ethan Marcotte (bit.ly/a02cFa). Responsive Web design is about using context clues from the browser to tailor the way a site is presented to a user, and even the way it behaves. It's about responding to information presented by the browser with a customized and compelling experience that fits a user's interaction expectations on that device. If the right clues reveal enough information about the browser and device in question, it becomes feasible to provide customizations, such as alternate images and fluid layouts.

Traditionally, you get those context clues by parsing the User Agent string presented by the browser—a practice called browser

Figure 1 **A Sample HTML Document**

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <link rel="stylesheet" type="text/css" href="media.css">
  </head>
  <body>
    <article id="small">This is a small screen</article>
    <article id="medium">This is a medium screen</article>
    <article id="large">This is a large screen</article>
    <article id="landscape">... and you're in landscape mode...</article>
  </body>
</html>
```

Figure 2 **CSS Media Queries Defined Inline with the @media Directive**

```
article {
  display: none;
}

body {
  font-size: 24px;
  font-weight: 800;
  color: white;
}

@media screen and (max-width: 480px) {
  body {
    background-color: #ff2a18;
  }

  #small {
    display: inline;
  }
}

@media screen and (min-width: 480px) and (max-width: 1024px) {
  body {
    background-color: #00ff00;
  }

  #medium {
    display: inline;
  }
}

@media screen and (min-width: 1024px) {
  body {
    background-color: #0000ff;
  }

  #large {
    display: inline;
  }
}

@media screen and (orientation: landscape) {
  body {
    background-color: #ff7f00;
  }

  #landscape {
    display: inline;
  }
}
```

or UA sniffing—and then using that information to determine how to proceed. I covered UA sniffing in my September 2011 article, "No Browser Left Behind: An HTML5 Adoption Strategy" (msdn.microsoft.com/magazine/hh394148). As I noted then, browser sniffing is unreliable and has essentially been replaced in favor of *feature detection*, which helps you make decisions about markup, CSS and JavaScript, based on available features, and not on a given browser and version.

In the context of responsive Web design, all modern desktop and major mobile browsers support a feature called *media queries*, a CSS3 module that extends the idea of media types introduced in CSS 2.1—the ability to specify alternative style sheets for print, screen and the like—with the ability to determine some of the physical characteristics of a device visiting a site. Similar to the rationale for using feature detection, media queries allow you to focus on getting the right context clues for a current visitor, which you can then use to responsively tailor the delivered experience. The power of media queries is that they provide such clues to your style sheets.

In this article, I'll provide an overview of the media queries CSS3 module. I'll briefly discuss its syntax and use, and then show a simple example that leverages media queries to build smartphone- and tablet-friendly views for an online photo gallery. Finally, I'll share a tip essential for all forms of mobile development before wrapping up with a brief discussion on media query listeners, a separate World Wide Web Consortium (W3C) specification (bit.ly/wyYBDG) that provides an API for reacting to changes in the media environment via JavaScript. By the end of this article, you should have a solid foundation for creating responsive Web sites and applications using only CSS and some tailored styles.

CSS Media Queries

As mentioned previously, CSS 2.1 currently supports media-dependent style sheets based on "media type" declarations (bit.ly/xhD3HD). The following markup illustrates how you can use media types to specify conditional style sheets:

```
<link rel="stylesheet" type="text/css" href="main.css" media="screen" />
<link rel="stylesheet" type="text/css" href="print.css" media="print" />
```

With these elements in place, a site will load a different style sheet from the default (or "screen") style sheet when a page is loaded in print preview mode. As useful as this feature is (though historically under-utilized), it doesn't provide any useful context clues to use for building truly responsive Web sites. And though the media types specification details the use of 10 media types, browser vendors never fully embraced all of the specified types.

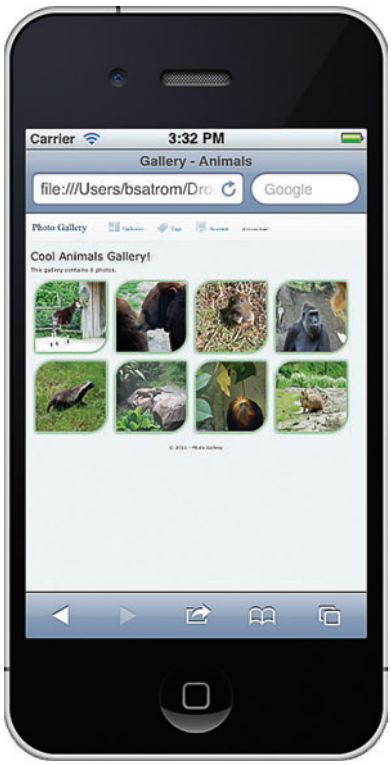


Figure 3 **Mobile View Without Media Queries**

The "media" attribute lays a solid foundation, though, and the W3C decided to build media queries (bit.ly/zbleDg) right on top. As you specify a media type (such as print, screen or all) media queries let you add expressions to the media attribute that check for the presence of certain media features, such as the width, height, orientation and even screen resolution of the current device. Here's an example that loads the main.css style sheet if the media type is screen and the device width is at least 800 pixels:

```
<link rel="stylesheet" media="screen and (min-width: 800px)" href="main.css" />
```

The media query is what's inside the parentheses. The left side specifies the property to test (width), with an optional min- or max- modifier, and the right side specifies the value of that property. If the device or browser in question presents a screen width of at least 800 pixels, the styles in main.css will be applied. If not, they won't. This illustrates how

Figure 4 **Modifying Styles for Mobile Devices**

```
body {
  min-width: 120px;
  max-width: 320px;
}

h1 {
  font-size: 1.5em;
}

img {
  width: 250px;
  height: 187.5px;
}

ul.thumbnails li {
  width: 265px;
    height: 200px;
    line-height: 200px;
}

ul.thumbnails li span.image-overlay {
    width: 265px;
}
```

media queries give developers useful context clues for creating responsive Web sites and applications.

Getting Started

As noted, CSS3 media queries are supported in the latest versions of all major browsers (bit.ly/wpamib), so you should feel comfortable using them today. There are three ways to leverage media queries for your sites. The first is to conditionally load entire style sheets, as shown in the previous example. The second method is to use @import directives in your style sheets. Here's the same test, this time as an import directive:

```
@import url("main.css") screen and (min-width: 800px);
```

Just as in the first example, this statement will evaluate the width of the current device and, if it's at least 800 pixels, load main.css and apply its styles.

Finally, you can use media queries inline in CSS using @media directives, like so:

```
@media screen and (min-width: 800px) { ... }
```

In this case, instead of defining styles in a separate file, you place them inline in an existing style sheet, and wrap them with a media query that ensures these styles are applied only when appropriate.

Figure 7 **Styles for Mobile-Friendly Navigation and Image Info**

```
#banner {
  height: 110px;
  background: #eaeaea;
}

#menu li {
  display: block;
  margin-top: 3px;
  margin-bottom: 3px;
}

#menu li.tags a,
#menu li.galleries a,
#menu li.account a {
  background: none;
}

#menu li.login {
  display: none;
}

ul.thumbnails li span.image-overlay {
  display:block;
  position: absolute;
  top: 0;
  left: 0;
  line-height: normal;
  width: 515px;
    padding: 5px;
}

ul.thumbnails li {
    background: #f3f6f7;
    border-color: #dbe2e5;
    border-radius: 7px;
    box-shadow: 0px 0px 20px 5px #A9A9A9;
}
```

Now let's look at media queries in action, starting with a simple example. If you want to try this at home, you can check out the sample source code for this article, or open up your favorite IDE or text editor and add the markup in **Figure 1**.

Next, create a style sheet called media.css in the same folder, and add the styles defined in **Figure 2**. Notice that I've defined all of my @media directives at the bottom of the file; you'll likely want to do the same in your style sheets. If you choose to define multiple style sheets in your markup, you'll also want to place <link> elements for specific style sheets after your main style sheets. Due to the cascading behavior of CSS, your media-specific rules will probably need to supersede your primary rules and should be defined appropriately. You'll notice that I've defined styles for my article and body elements, and then followed these with media queries (and their associated rules) for each experience I wish to support. Screens smaller than 480 pixels will receive one set of additional styles; screens between 480 pixels and 1024 pixels will receive another; and screens larger than 1024 pixels receive a third set of styles. I've also added a query for orientation, and will apply yet another set of styles if the device is in landscape mode.

Once you've created the page in **Figure 1** and style sheet in **Figure 2**, open the page

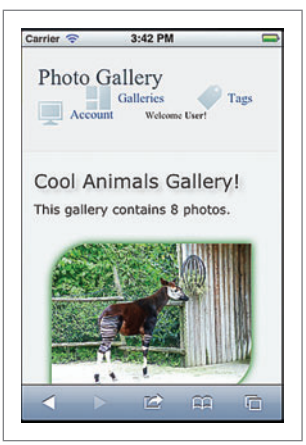


Figure 5 **Mobile View with Improved Images**

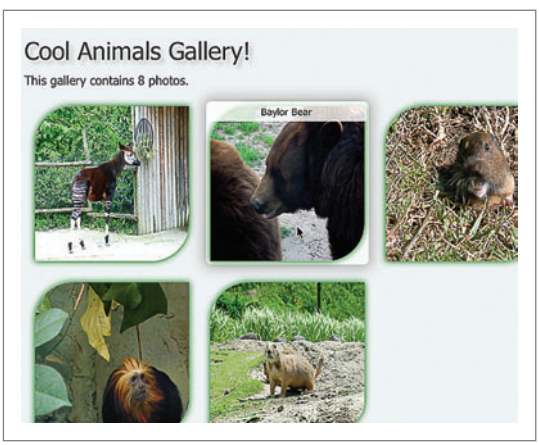


Figure 6 **An Image with the Hover Effect Applied**

in any browser. Testing your media queries is simple; you can even do so without a mobile device. Just resize your browser window and watch how the text and background colors change as the context of the visible window changes.

Now that we've laid a foundation for media queries, let's take things a step further and walk through a more realistic example. For this example I'm going to add media queries to an online photo gallery so I can support smartphones and tablets. You'll find the code for this example, which uses the Photo Gallery template that ships with WebMatrix (webmatrix.com), at code.msdn.microsoft.com/mag201204HTML5.

If you open this page in a mobile or tablet browser, you'll notice the page appears "zoomed out" and is difficult to see or read, as **Figure 3** shows. With a few media queries, and some other tricks, we can create a tailored experience without new markup or browser sniffing of any kind.

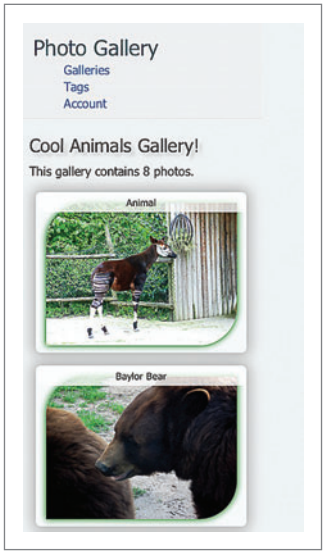


Figure 8 **The Gallery with Better Navigation and Image Boxes**

At the bottom of the desktop.css file created for this sample, start by adding your media query directive and condition. Because my current concern is the smallest of screens, I'll create the following definition:

```
@media screen and (max-width: 480px) {}
```

Now, inside that definition, let's create some rules to resize and reposition the images for smaller screens. Those rules are shown in **Figure 4**.

If you refresh the page after adding these styles, you should see something similar to what's shown in **Figure 5**. The images look better, but the main navigation is misaligned. What's more, the gallery uses a CSS :hover pseudo-selector to display additional information about each image, as shown in **Figure 6**. Because this is a user experience that doesn't make sense in an environment where hover events are less common (smartphones and tablets), we need to come up with an alternative presentation for that aspect of the page as well. We'll do so by adding the styles in **Figure 7** inside the current @media directive.

Figure 9 **Styles for the Tablet-Friendly View**

```
body {
  min-width: 480px;
  max-width: 800px;
}

img {
  width: 500px;
  height: 375px;
  align: center;
}

ul.thumbnails li {
  width: 515px;
    height: 390px;
    line-height: 200px;
}
```

Now, if you refresh the page in your browser, you should see cleaned-up navigation, along with the information that previously appeared on a hover event, similar to **Figure 8**.

So far, we have a desktop view for our gallery, as well as a decent mobile version, and all we needed were a few additional CSS rules. Let's take this a step further and add support for screens larger than 480 pixels and smaller than 1024 pixels—tablets.

If you're viewing the photo gallery in a desktop browser and you resize to a size smaller than 1024 pixels (the screen shown in **Figure 8**), you'll notice that many of the images are cut off. Because tablet devices generally have a larger screen size, it makes sense to provide larger images on the gallery page. So let's add another media query for tablet-size devices:

```
@media screen and (min-width:480px) and (max-width:1024px)
```

In the online source, I've refactored some of the smartphone rules into a set of styles that apply to all non-desktop devices, so all I need to set for my tablet view are those styles that relate to document, image and thumbnail size, as shown in **Figure 9**.

Now refresh the screen in a tablet-view browser, and you'll see something similar to **Figure 10**. We're now targeting multiple screen

sizes—and thus multiple devices—and all it cost us was the time to create some alternative styles.

Mobile First

Before I wrap up this discussion on media queries, I want to share a tip that isn't specific to the use of media queries, but is important when doing mobile Web development of any kind. In particular, you should set the viewport and design for mobile *first*.

If you've been coding along as I've walked through media queries in this article, you might have seen a slight issue when viewing the page on a smartphone or smartphone emulator: On some devices, the "zoomed out" effect didn't change, even after the conditional rules were applied. What's happening is that the mobile browser is trying to "optimize" the experience on a smaller screen, which is helpful for sites that pay no attention to mobile users. For this site, however, the preference is that this optimization doesn't occur, and you can override this behavior by adding a single <meta> tag to the <head> of your page, like so:

```
<meta name="viewport" content="width=device-width">
```

Now, if you refresh the page, and your media queries are properly defined, you should see your rules in effect.

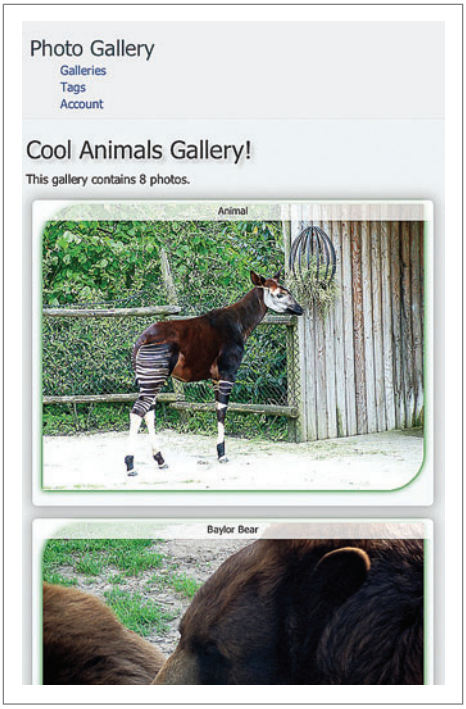


Figure 10 **Photo Gallery, Tablet-Friendly View**

> Mobile first is about making your mobile experience a top priority.

You might have also taken note of the fact that, in the photo gallery example, I added a mobile experience to an existing site. As a developer familiarizing yourself with media queries, you're likely do this initially as well. However, there's a growing movement that emphasizes a "mobile first" approach to designing and developing Web sites and applications. Just as it sounds, mobile first is about making your mobile experience a top priority, instead of an afterthought (as was often done in the past). When you consider that in the last two years, smartphone and tablet sales have outstripped PC sales and that mobile browsing now makes up more than half of all Web browsing, this makes sense. Beyond that, a mobile-first mindset encourages "progressive enhancement," whereby sites are built from a baseline (and often text-only) experience, and progressively enhanced for more capable browsers and devices. To learn more about the mobile-first approach, I highly recommend the book by that name, by Luke Wroblewski (available at bit.ly/zd9UWT).

Figure 11 **Working with Media Query Listeners**

```
listener = window.msMatchMedia("(min-width: 480px)");
evaluate(listener); // Perform an initial evaluation

listener.addListener(evaluate); // Evaluate each time the width changes

 function evaluate(listener) {
   if (mql.matches) {
    expandCommentList(); // Screen is at least 480px
   } else {
     shrinkCommentList(); // Screen is smaller
   }
 }
```

Media Query Listeners

As an extension to media queries support in CSS3, the CSS Working Group is looking to add JavaScript APIs that enable developers to evaluate media queries at run time and listen for runtime changes in media query conditions. These APIs are documented in the CSSOM View Module specification (bit.ly/w8Ncq4). **Figure 11** shows an example of performing media queries and creating listeners from within JavaScript.

Listeners have the potential to be quite valuable for cases where a device's viewport might change at run time—for instance, when the orientation changes on smartphones and tablets. With media query listeners, you can respond to these changes, not just from within CSS, but from your scripts. The CSSOM View specification is currently in a Working Draft state, but Internet Explorer 10 Platform Preview provides support for media query listeners, so you can try this out at Internet Explorer Test Drive (bit.ly/gQk7wZ).

The ability to support multiple platforms, screens and experiences in Web applications has come a long way over the last several years. Whereas in the past it was normal to create device-specific versions of a site, the proliferation of mobile-friendly devices has quickly made device targeting unfeasible. Thankfully, the introduction of CSS3 media queries across all major desktop and mobile browsers means you can now use the context clues that really matter to deliver tailored experiences for a wide range of devices, all with conditional CSS. For more information on CSS3 media queries, I recommend the chapter on media queries from Peter Gasston's excellent "The Book of CSS3" (No Starch Press, 2011). For a gallery of great experiences delivered with the help of media queries, check out the media queries site. I hope this article has equipped you with enough information to get started building tailored experiences today. I look forward to seeing the awesome experiences you create! ■

Brandon Satrom *is the product manager for Kendo UI (kendoui.com), an HTML5 and mobile toolset from Telerik. He blogs at userinexperience.com and can be followed on Twitter at twitter.com/BrandonSatrom.*

Thanks to the following technical experts for reviewing this article: *John Box, Sharon Newman and Jacob Rossi*

A Code-Based Introduction to C++ AMP

Daniel Moth

Visual Studio 11 brings support for heterogeneous computing to the mainstream through a technology called C++ Accelerated Massive Parallelism (C++ AMP). This enables you to take advantage of accelerators, such as GPUs, for speeding up data parallel algorithms.

C++ AMP delivers performance, in a hardware-portable manner, without compromising the productivity you've come to expect from modern C++ and the Visual Studio package. It could provide orders of magnitude speed gains compared with just using the CPU only. At conferences I typically demo a single process taking advantage of both NVIDIA and AMD GPUs at the same time, while still having a CPU fallback solution.

In this code-driven introduction that explores C++ AMP, I'll assume that you're reading every line of code in the article. The inline code is a core part of the article, and what's in the C++ code won't necessarily be repeated in the article's text.

Setup and Example Algorithm

First, let's understand the simple algorithm that we'll be working with along with the necessary setup code, in preparing for the conversion to using C++ AMP later.

This article covers a prerelease technology called C++ AMP that will ship with Visual Studio 11. All information is subject to change.

This article discusses:

- Setting up a sample algorithm
- Converting the algorithm to C++ AMP
- Using array_view, extent, index, restrict(amp) and parallel_for_each

Technologies discussed:

C++ AMP, Visual Studio 11

Code download available at:

code.msdn.microsoft.com/mag201204CPPAMP

Create an empty C++ project, add a new empty C++ file (Source.cpp) and type in the following self-explanatory code (I'm using noncontiguous line numbers to facilitate explanation in the article text, and you'll find the same line numbers in the accompanying downloadable project):

```
1 #include <amp.h>                // C++ AMP header file
3 #include <iostream>             // For std::cout etc
4 using namespace concurrency;    // Save some typing :)
5 using std::vector;     // Ditto. Comes from <vector> brought in by amp.h
6
79 int main()
80 {
81   do_it();
82
83   std::cout << "Hit any key to exit..." << std::endl;
84   std::cin.get();
85 }
```

C++ AMP introduces a number of types across a number of header files. As per lines 1 and 4 in the preceding code snippet, the main header file is *amp.h* and the main types are added to the existing concurrency namespace. No additional setup or compilation option is required for using C++ AMP. Now let's add a do_it function above main (see **Figure 1**).

In lines 59, 60 and 68, the code uses std::vector objects as flat containers for each matrix, even though a two-dimensional type is what you'd really want to be dealing with—more on this later.

It's important to understand the usage of the lambda expressions on lines 64 and 65 that are passed to the std::generate method for populating the two vector objects. This article assumes you can use lambdas in C++ proficiently. For example, you should instantly understand that if the variable i was captured by value (by modifying the capture list either like this *[i]* or like this *[=]* and using the mutable keyword) then every member of the vector would have been initialized to 0! If you aren't comfortable with using lambdas (a wonderful addition to the C++ 11 standard), please read the MSDN Library article "Lambda Expressions in C++" (msdn.microsoft.com/library/dd293608) and come back here when you're done.

The do_it function introduced a call to perform_calculation, which is coded as follows:

```
7  void perform_calculation(
8    vector<int>& vA, vector<int>& vB, vector<int>& vC, int M, int N)
9  {
15   for (int i = 0; i < M; i++)
16   {
17     for (int j = 0; j < N; j++)
18     {
19       vC[i * N + j] = vA[i * N + j] + vB[i * N + j];
20     }
22   }
24 }
```

In this simplistic matrix addition example, one thing that sticks out is that the multidimensionality of the matrix is lost because of the linearized storage of the matrix in a vector object (which is why you had to pass in the matrix dimensions along with the vector objects). Furthermore, you have to perform funny arithmetic with the indices on line 19. This point would've been even more obvious if you wanted to add sub-matrices of these matrices together.

There's been no C++ AMP code so far. Next, by changing the perform_calculation function, you'll see how you can start introducing some of the C++ AMP types. In later sections you'll learn how to fully leverage C++ AMP and speed up your data parallel algorithms.

array_view<T, N>, extent<N> and index<N>

C++ AMP introduces a concurrency::array_view type for wrapping containers of data—you can think of it as a smart pointer. It represents data in a rectangular manner, contiguous in the least-significant dimension. The reason for its existence will become obvious later, and next you'll see some aspects of its usage. Let's change the body of the perform_calculation function as follows:

```
11     array_view<int> a(M*N, vA), b(M*N, vB);
12     array_view<int> c(M*N, vC);
14
15     for (int i = 0; i < M; i++)
16     {
17       for (int j = 0; j < N; j++)
18       {
19         c(i * N + j) = a(i * N + j) + b(i * N + j);
20       }
22     }
```

This function, which compiles and runs on the CPU, has the same output as before. The only difference is the gratuitous use of the array_view objects that are introduced on lines 11 and 12. Line 19 still has the funny indexing (for now), but now it's using the array_view objects (a, b, c) instead of the vector objects (vA, vB and vC) and it's accessing the elements through the array_view function operator (in contrast with the prior usage of the vector subscript operator—more on this later).

You must tell the array_view through a template argument (int in this example) the element type of the container it wraps; you'll pass the container as the last constructor argument (for example, the vC variable of vector type on line 12). The first constructor argument is the number of elements.

You can also specify the number of elements with a concurrency::extent object, so you can change lines 11 and 12 as follows:

```
10     extent<1> e(M*N);
11     array_view<int, 1> a(e, vA), b(e, vB);
12     array_view<int, 1> c(e, vC);
```

The extent<N> object represents a multidimensional space, where the rank is passed as a template argument. The template argument is 1 in this example, but the rank can be any value greater than zero. The extent constructor accepts the size of each dimension that the extent object represents, as shown on line 10. The extent object can then be passed to the array_view object constructor to define its shape, as shown on lines 11 and 12. On those lines I also added a second template argument to the array_view indicating that it represents a one-dimensional space—as in the earlier code example, I could've safely omitted that because 1 is the default rank.

Now that you know of these types, you can make further modifications to the function so it can access the data in a more natural two-dimensional manner, which resembles the matrix world more closely:

```
10     extent<2> e(M, N);
11     array_view<int, 2> a(e, vA), b(e, vB);
12     array_view<int, 2> c(e, vC);
14
15     for (int i = 0; i < e[0]; i++)
16     {
17       for (int j = 0; j < e[1]; j++)
18       {
19         c(i, j) = a(i, j) + b(i, j);
20       }
22     }
```

The changes to lines 10-12 make the array_view objects two-dimensional, so we'll require two indices to access an element. Lines 15 and 17 access the bounds of the extent through its subscript operator instead of directly using the variables M and N; once you've encapsulated the shape into the extent, you can now use that object throughout your code.

The important change is on line 19, where you no longer need funny arithmetic. The indexing is much more natural, making the entire algorithm itself much more readable and maintainable.

If the array_view was created with a three-dimensional extent, then the function operator would expect three integers in order to access an element, still from the most-significant to the least-significant dimension. As you might expect from a multidimensional API, there's also a way to index into an array_view through a single object passed to its subscript operator. The object must be of type concurrency::index<N> where N matches the rank of the extent with which the array_view is created. You'll see later how index objects can be passed to your code, but for now let's create one manually in order to get a feel for them and see them in action, by modifying the function body as follows:

```
10     extent<2> e(M, N);
11     array_view<int, 2> a(e, vA), b(e, vB);
12     array_view<int, 2> c(e, vC);
13
14     index<2> idx(0, 0);
15     for (idx[0] = 0; idx[0] < e[0]; idx[0]++)
16     {
17       for (idx[1] = 0; idx[1] < e[1]; idx[1]++)
18       {
19         c[idx] = a[idx] + b[idx];
//19           //c(idx[0], idx[1]) = a(idx[0], idx[1]) + b(idx[0], idx[1]);
20       }
22     }
```

As you can see from lines 14, 15, 17 and 19, the concurrency::index<N> type has a very similar interface to the extent type, except index represents an N-dimensional point instead of an N-dimensional space. Both the extent and index types support a number of arithmetic operations through operator overloading—for example, the increment operation shown in the preceding example.

Previously the loop variables (i and j) were used to index into the array_view, and now they can be replaced with a single index object on line 19. This demonstrates how, by using the array_view subscript operator, you can index into it with a single variable (in this example, idx of type index<2>).

Figure 1 **The do_it Function, Called from main**

```
52 void do_it()
53 {
54   // Rows and columns for matrix
55   const int M = 1024;
56   const int N = 1024;
57
58   // Create storage for a matrix of above size
59   vector<int> vA(M * N);
60   vector<int> vB(M * N);
61
62   // Populate matrix objects
63   int i = 0;
64   std::generate(vA.begin(), vA.end(), [&i](){return i++;});
65   std::generate(vB.begin(), vB.end(), [&i](){return i--;});
66
67   // Output storage for matrix calculation
68   vector<int> vC(M * N);
69
70   perform_calculation(vA, vB, vC, M, N);
76 }
```

At this point you have a basic understanding of three new types introduced with C++ AMP: array_view<T,N>, extent<N> and index<N>. They have more to offer, as shown in the class diagrams in **Figure 2**.

The real power and motivation for using this multidimensional API is to execute your algorithms on a data parallel accelerator, such as the GPU. To do that you need an entry point in the API for executing your code on the accelerator, plus a way to check at compile time that you're using a subset of the C++ language that can be executed efficiently on such an accelerator.

parallel_for_each and restrict(amp)

The API that instructs the C++ AMP runtime to take your function and execute it on the accelerator is a new overload to concurrency::parallel_for_each. It accepts two arguments: an extent object and a lambda.

The extent<N> object, which you're already familiar with, is used to determine how many times the lambda will be called on the accelerator, and you should assume that each time it will be a separate thread calling your code, potentially concurrently, without any order guarantees. For example, an extent<1>(5) will result in five calls to the lambda you pass to the parallel_for_each, whereas an extent<2>(3,4) will result in 12 calls to the lambda. In real algorithms you'll typically be scheduling thousands of calls to your lambda.

The lambda must accept an index<N> object, which you're already familiar with. The index object must have the same rank as the extent object passed to parallel_for_each. The index value will be different each time your lambda is called, of course—that's how you distinguish between two different invocations of your lambda. You could think of the index value as the thread ID.

Here's a code representation of what I described so far with parallel_for_each:

```
89     extent<2> e(3, 2);
90     parallel_for_each(e,
91       [=](index<2> idx)
92       {
93         // Code that executes on the accelerator.
94         // It gets invoked in parallel by multiple threads
95         // once for each index "contained" in extent e
96         // and the index is passed in via idx.
97         // The following always hold true
98         //      e.rank == idx.rank
99         //      e.contains(idx) == true
100        //      the function gets called e.size() times
101        // For this two-dimensional case (.rank == 2)
102        //      e.size() == 3*2 = 6 threads calling this lambda
103        // The 6 values of idx passed to the lambda are:
104        //      { 0,0 } { 0,1 } { 1,0 } { 1,1 } { 2,0 } { 2,1 }
105      }
106    );
107    // Code that executes on the host CPU (like line 91 and earlier)
```

This simple code, without an important addition to line 91, won't compile:

```
error C3577: Concurrency::details::_Parallel_for_each argument #3 is illegal:
missing public member: 'void operator()(Concurrency::index<_Rank>) restrict(amp)'
```

As the code was written, there would be nothing stopping you from using in the lambda body (lines 92-105) anything that the full C++ language (as supported by the Visual C++ compiler) allows. However, you're restricted from using certain aspects of the C++ language on current GPU architectures, so you must indicate which parts of your code should comply with these restrictions (so you can find out at compile time if you're breaking any rules). The indication must be made on the lambda and on any other function signatures that you call from the lambda. So you must modify line 91 as follows:

```
91        [=](index<2> idx) restrict(amp)
```

This is a key new language feature of the C++ AMP specification that's added to the Visual C++ compiler. Functions (including lambdas) can be annotated with restrict(cpu), which is the implicit default, or restrict(amp) as shown in the previous code sample, or with a combination—for example, restrict(cpu, amp). No other options exist. The annotation becomes part of the function signature so it participates in overloading, which was a key motivation when designing it. When a function is annotated with restrict(amp), it's checked against a set of restrictions, and if one is violated you receive a compiler error. The full set of restrictions is documented in this blog post: bit.ly/vowVIV.

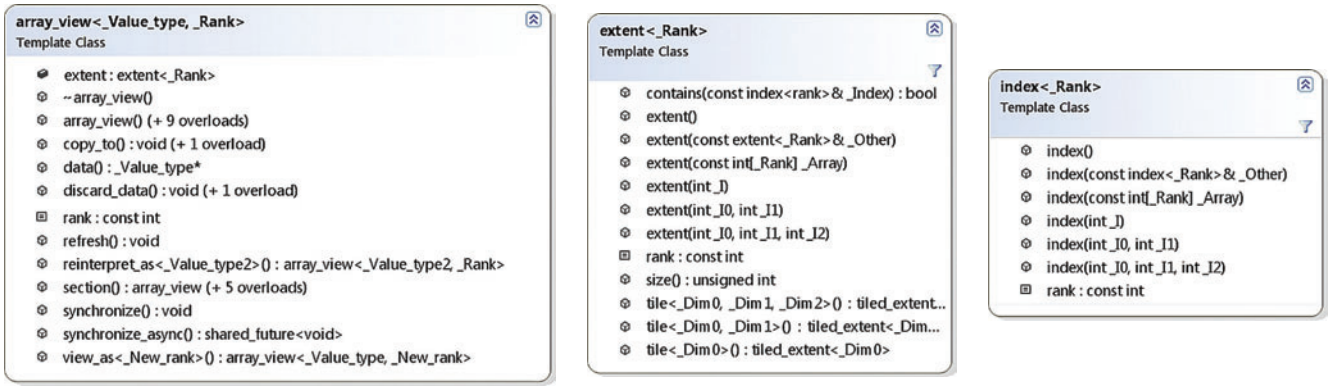


Figure 2 **array_view, extent and index Classes**

One of the restrict(amp) restrictions for lambdas is that they can't capture variables by reference (see caveat near the end of this article), nor can they capture pointers. With that restriction in mind, when you look at the last code listing for parallel_for_each, you would be right in wondering: "If I can't capture by reference and can't capture pointers, how will I ever observe results—that is, desirable side effects—from the lambda? Any changes I make to variables that I capture by value won't be available to the outer code once the lambda completes."

The answer to that question is a type that you already know: array_view. The array_view object is allowed to be captured by value in the lambda. It's your mechanism for passing data in and out. Simply use array_view objects to wrap the real containers, then capture the array_view objects in the lambda for accessing and populating, and then access the appropriate array_view objects after the call to parallel_for_each.

Bringing It All Together

With your new knowledge, you can now revisit the earlier serial CPU matrix addition (the one that used array_view, extent and index) and replace lines 15-22 as follows:

```
15    parallel_for_each(e, [=](index<2> idx) restrict(amp)
16    {
19      c[idx] = a[idx] + b[idx];
22    });
```

You see that line 19 remained the same, and the double nested loop with manual index object creation within the extent bounds is replaced with a call to the parallel_for_each function.

When working with discrete accelerators that have their own memory, the capture of the array_view objects in the lambda passed to parallel_for_each results in a copy of the underlying data to the accelerator's global memory. Similarly, after the parallel_for_each call, when you access the data through the array_view object (in this example, through c) the data gets copied back to the host memory from the accelerator.

You should know that if you want to access the results from array_view c through the original container vC (and not through the array_view), then you should call the synchronize method of the array_view object. The code would work as it stands because the array_view destructor calls synchronize on your behalf, but any exceptions would get lost that way, so I advise you to call synchronize explicitly instead. So add a statement anywhere after the parallel_for_each call, like this:

```
23          c.synchronize();
```

The reverse (ensuring that the array_view has the latest data from its original container if it has changed) is achieved via the refresh method.

More importantly, copying data across (typically) a PCIe bus can be very expensive, so you only want to copy data in the direction necessary. On the earlier listing you can modify lines 11-13 to indicate that the underlying data of array_view objects a and b must be copied to the accelerator (but won't be copied back), and also that the underlying data of array_view c need not be copied to the accelerator. The necessary changes are bolded in the following snippet:

```
11          array_view<const int, 2> a(e, vA), b(e, vB);
12          array_view<int, 2> c(e, vC);
13          c.discard_data();
```

Figure 3 **Picking an Accelerator**

```
26 accelerator pick_accelerator()
27 {
28   // Get all accelerators known to the C++ AMP runtime
29   vector<accelerator> accs = accelerator::get_all();
30
31   // Empty ctor returns the one picked by the runtime by default
32   accelerator chosen_one;
33
34   // Choose one; one that isn't emulated, for example
35   auto result =
36     std::find_if(accs.begin(), accs.end(), [] (accelerator acc)
37   {
38     return !acc.is_emulated; //.supports_double_precision
39   });
40   if (result != accs.end())
41     chosen_one = *(result); // else not shown
42
43   // Output its description (tip: explore the other properties)
44   std::wcout << chosen_one.description << std::endl;
45
46   // Set it as default ... can only call this once per process
47   accelerator::set_default(chosen_one.device_path);
48
49   // ... or just return it
50   return chosen_one;
51 }
```

However, even with these modifications in place, the matrix addition algorithm isn't sufficiently compute-intensive to offset the overhead of copying the data, so it's in fact not a good candidate for parallelization with C++ AMP. I've been using it only to teach you the basics!

Having said that, by using this simple example throughout the article, you now have the skills to parallelize other algorithms that are compute-intensive enough to yield benefits. One such algorithm is matrix multiplication. Without any commentary from me, please ensure that you understand this simple serial implementation of the matrix multiplication algorithm:

```
void MatMul(vector<int>& vC, const vector<int>& vA,
  const vector<int>& vB, int M, int N, int W)
{
  for (int row = 0; row < M; row++)
  {
    for (int col = 0; col < N; col++)
    {
      int sum = 0;
      for(int i = 0; i < W; i++)
        sum += vA[row * W + i] * vB[i * N + col];
      vC[row * N + col] = sum;
    }
  }
}
```

… and the corresponding C++ AMP implementation:

```
array_view<const int, 2> a(M, W, vA), b(W, N, vB);
array_view<int, 2> c(M, N, vC);
c.discard_data();
parallel_for_each(c.extent, [=](index<2> idx) restrict(amp)
{
  int row = idx[0]; int col = idx[1];
  int sum = 0;
  for(int i = 0; i < b.extent[0]; i++)
    sum += a(row, i) * b(i, col);
  c[idx] = sum;
});
c.synchronize();
```

On my laptop machine the C++ AMP matrix multiplication yields a performance improvement of more than 40 times compared with the serial CPU code for M=N=W=1024.

Now that you understand all the basics, you might be wondering how you can choose which accelerator to execute your algorithm on, once you've implemented it using C++ AMP. Let's examine that next.

Kendo UI

THE ART OF WEB DEVELOPMENT

Everywhere You Want It

Desktop and Mobile. Keyboard and Touch. It's all there.

Kendo UI is designed to work with all forms of input, from traditional keyboard and mouse to modern touchscreens. Using one toolset, you get everything you need to build apps and sites that run in desktop browsers, tablets, and mobile devices. One toolset. One app. Unlimited reach.

Download the future of JavaScript development at **www.kendoui.com** or scan

accelerator and accelerator_view

Part of the concurrency namespace is the new accelerator type. It represents a device on the system that the C++ AMP runtime can use, which for the first release is hardware with a DirectX 11 driver installed (or DirectX emulators).

When the C++ AMP runtime starts, it enumerates all accelerators and, based on internal heuristics, picks one as the default. That's why you didn't have to deal with accelerators directly in all the preceding code—there was a default picked for you. If you want to enumerate the accelerators and even choose the default yourself, it's very easy to do, as the self-explanatory code in **Figure 3** shows.

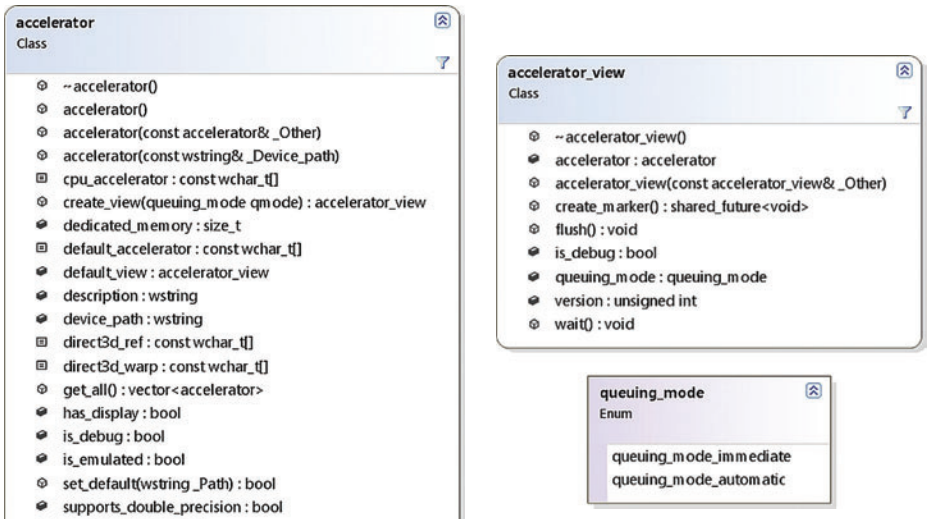


Figure 4 **accelerator and accelerator_view Classes**

On line 38 you can see querying of one of the many accelerator properties, and others are shown in **Figure 4**.

If you want to have different parallel_for_each calls that use different accelerators, or for whatever other reason you want to be more explicit than setting the default accelerator globally for a process, then you need to pass an accelerator_view object to the parallel_for_each. This is possible because parallel_for_each has an overload that accepts an accelerator_view as the first parameter. Obtaining an accelerator_view object is as easy as calling default_view on an accelerator object; for example:

```
accelerator_view acc_vw = pick_accelerator().default_view;
```

Beyond DirectX 11 hardware, there are three special accelerators that C++ AMP makes available:

- direct3d_ref: useful for correctness debugging, but not useful in production as it's extremely slower than any real hardware.
- direct3d_warp: a fallback solution for executing your C++ AMP code on the CPU using multi-core and Streaming SIMD Extensions today.
- cpu_accelerator: not capable of executing C++ AMP code at all, in this release. It's only useful for setting up staging arrays (an advanced optimization technique), which is beyond the scope of this article but is described in this blog post: bit.ly/vRksnn.

Tiling and Further Reading on Your Own

The most important topic not covered in this article is tiling.

From a scenario perspective, tiling takes the orders of magnitude performance gains that you see with the coding techniques explored so far and (potentially) makes the gains even larger. From an API perspective, tiling consists of tiled_index and tiled_extent types, as well as a tile_barrier type and a tile_static storage class. There's also an overload to parallel_for_each that accepts a tiled_extent object, and whose lambda accepts a tiled_index object. Within that lambda you're allowed to use tile_barrier objects and tile_static variables. I cover tiling in my second C++ AMP article on p. 40.

There are other topics you can explore on your own with the help of blog posts and the online MSDN documentation:

- <amp_math.h> is a math library with two namespaces, one for high-precision math functions and the other for fast but less-accurate math functions. Opt in based on your hardware capabilities and your scenario requirements.
- <amp_graphics.h> and <amp_short_vectors.h> plus some DirectX interop functions are available for working with graphics programming.
- concurrency::array is a container data type that's bound to an accelerator, with a nearly identical interface to array_view. This type is one of two types (the other being texture in the graphics namespace) that must be captured by reference in the lambda passed to parallel_for_each. This is the caveat I referred to earlier in the article.
- Support for DirectX intrinsics such as atomics for cross-thread synchronization.
- GPU debugging and profiling in Visual Studio 11.

Future-Proofing Your Investments

In this article I introduced you to a modern C++ data parallel API that allows you to express your algorithms in a manner that enables your app to utilize GPUs for performance gains. C++ AMP is designed so it can future-proof your investments for hardware that we've yet to see.

You learned how a handful of types (array_view, extent and index) help you work with multidimensional data, combined with a single global function (parallel_for_each) that allows you to execute your code—starting from a restrict(amp) lambda—on an accelerator (that you can specify through accelerator and accelerator_view objects).

Beyond the Microsoft Visual C++ implementation, C++ AMP is provided to the community as an open specification that anyone can implement, on any platform. ■

DANIEL MOTH *is a principal program manager in the Microsoft Developer Division. He can be reached via his blog at danielmoth.com/Blog.*

THANKS to the following technical experts for reviewing this article: *Steve Deitz, Yossi Levanoni, Robin Reynolds-Haertle, Stephen Toub and Weirong Zhu*

Kendo UI

THE ART OF WEB DEVELOPMENT

Everything You Need

For JavaScript & HTML5 development

Kendo UI is a complete, integrated package that provides developers a jQuery-based toolset which includes a powerful data source, dynamic data visualizations, and blazing fast micro-templates, all backed by industry leading professional support. Download Kendo UI and start building amazing sites and apps today.

Download the future of JavaScript development at **www.kendoui.com** or scan

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

YOUR MAP TO THE .NET DEVELOPMENT PLATFORM

BIG CODE IN THE BIG APPLE

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

PLATINUM SPONSOR SUPPORTED BY

esri Microsoft Visual Studio msdn Visual Studio MAGAZINE

Register Before April 11 and Save $200!

Use Promo Code APRAD

Bright Lights, Big City!

Visual Studio Live! is thrilled to be back in New York! Join developers, software architects and designers in Brooklyn for 4 days of unbiased and cutting-edge education on the Microsoft Platform.

"Excellent talks and very knowledgeable speakers. Cutting-edge information was presented and old concepts were reinforced. Great conference!"

Chris Dundon, Visual Studio Programmer, ISI Telemanagement Solutions

"As a one-man development team, using blogs and coding sites only help so much. Being able to get real-life training and talk to people who work in the industry are important. VSLive has offered a good opportunity to do just that, with a good range of technology areas and interests."

Kevin Dietrich, Owner/Developer, Kdietrich LLC

vslive.com/newyork

Scan this code to register and learn more about what Visual Studio Live! New York has to offer.

Brooklyn, NY

May 14-17

NY Marriott at the Brooklyn Bridge

PRODUCED BY

1105 MEDIA

MEDIA SPONSOR

THE CODE PROJECT WWW.CODEPROJECT.COM

Check Out What You'll Learn at Visual Studio Live! New York:

Track: Cloud Computing

Session: Building Scalable Apps in Windows Azure

You will learn:

- Patterns for making cloud applications scalable
- Patterns for making cloud based applications fault tolerant
- Cost-based design

Track: Cross Platform Mobile

Session: W16 ASP.NET MVC for Mobile Devices

You will learn:

- How to create an MVC3 application capable of targeting multiple device types
- How MVC4 enables multi-platform targeting
- How to leverage Razor layouts and views for various display types

Track: Data Management

Session: T-SQL Enhancements in SQL Server 2012

You will learn:

- New T-SQL features in SQL Server 2012
- The 22 new T-SQL functions, including 8 new analytic windowing functions
- OVER, THROW, OFFSET, metadata discovery, and contained databases

Track: HTML5

Session: Upgrade Your Website to HTML5

You will learn:

- How to upgrade your current website with HTML5
- How to use advanced HTML5 APIs which gracefully degrade
- How to enhance your website with the latest HTML5 goodies

Track: Silverlight / WPF

Session: Building Extensible XAML Client Apps

You will learn:

- How to leverage MEF, MVVM, and Prism to build extensible applications
- How to compose parent-child relationships in views so that you can plug in new user interface elements without the parent being coupled to the child views
- How to add whole new sets of functionality by plugging them in with Prism modules and regions

Track: Visual Studio 2010+ / .NET4+

Session: What's New in the .NET 4.5 BCL

You will learn:

- An understanding of the new APIs available to .NET developers
- What's been changed in 4.5 that may affect current code
- How to use this knowledge to move effectively into the next version of .NET

Track: Web

Session: Fast, Faster ... Async ASP.NET

You will learn:

- Load Testing ASP.NET Applications
- Asynchronous operations in ASP.NET WebForms
- Asynchronous operations in ASP.NET MVC
- How to use the new async/await in Web Applications

Track: Windows 8 / WinRT

Session: Lap Around the Windows Runtime

You will learn:

- An overview of the Windows Runtime in Windows 8
- How to use the Windows Runtime from .NET, HTML + JS, and C++
- Comparisons to existing frameworks and discussion of how to port existing code

Track: Windows Phone 7

Session: Building Windows Phone Applications Using the Fast Lane: Using RIA Services on Azure as Your Backend

You will learn:

- How you can use RIA services as the service layer beyond the Silverlight client
- How you can secure your service and push it to the cloud
- How easy it is to build your RIA service in Azure and hence scale with the success of your phone app

Register at vslive.com/newyork
Use Promo Code APRAD

VISUAL STUDIO LIVE! NEW YORK AGENDA AT-A-GLANCE

HTML5	Web	Visual Studio 2010+/.NET 4+	Cloud Computing	Data Management	Silverlight / WPF	Windows 8 / WinRT	Windows Phone 7	Cross Platform Mobile

Visual Studio Live! Pre-Conference Workshops: Monday, May 14, 2012 *(Separate entry fee required)*

WK1 **Workshop: Creating Today's User Experiences - An Entry Point for Developers** *Billy Hollis*	WK2 **Workshop: SQL Server for Developers** *Andrew Brust & Leonard Lobel*	WK3 **Workshop: Full Application Lifecycle with TFS and CSLA .NET** *Rockford Lhotka & Brian Randell*

Visual Studio Live! Day 1: Tuesday, May 15, 2012

Keynote *Microsoft To Be Announced*

T1 **A Lap Around WPF 4.5** *Pete Brown*	T2 **Microsoft To Be Announced**	T3 **HTML5/jQuery On-Ramp** *Rich Dudley*	T4 **Microsoft To Be Announced**
T5 **WPF Validation - Techniques & Styles** *Miguel Castro*	T6 **Get Started with XAML Development on Windows 8** *Pete Brown*	T7 **Upgrade Your Website to HTML5** *Rajesh Lal*	T8 **The LINQ Programming Model** *Marcel de Vries*

Lunch

T9 **What's New and Cool in Silverlight 5** *Pete Brown*	T10 **Building Windows 8 Applications with HTML5 and jQuery** *Rich Dudley*	T11 **HTML5** *Marcel de Vries*	T12 **Microsoft To Be Announced**
T13 **Infinite Whitespace: Implementing Viewport Navigation in XAML** *Billy Hollis*	T14 **Lap Around the Windows Runtime** *Rockford Lhotka*	T15 **Microsoft To Be Announced**	T16 **Microsoft To Be Announced**
T17 **Building Extensible XAML Client Apps** *Brian Noyes*	T18 **A look at Windows 8 Metro Apps and WinRT Internals** *Vishwas Lele*	T19 **Microsoft To Be Announced**	T20 **Developing Apps for Nokia Windows Phone** *Rajesh Lal*

Exhibitor Reception

Visual Studio Live! After Dark

Visual Studio Live! Day 2: Wednesday, May 16, 2012

Keynote: The Developer of the Future

Gabriel Torok, CEO PreEmptive Solutions and Justin Marks, Senior Program Manager for Microsoft's Team Foundation Server

W1 **Get Connected with Kinect** *Brian Peek*	W2 **Build Portable XAML Client App Logic and Resources** *Brian Noyes*	W3 **Windows Azure Platform Overview** *Vishwas Lele*	W4 **Building Windows Phone Applications Using the Fast Lane: Using RIA services on Azure As Your Backend** *Marcel de Vries*
W5 **Securing and Personalizing Silverlight 5 Client Apps** *Brian Noyes*	W6 **Building Metro Style Apps: Getting the UX Right** *Brian Randell*	W7 **Building Scalable Apps in Windows Azure** *Vishwas Lele*	W8 **Visual Studio for Mobile Apps on iOS, Android and WP7** *Miguel de Icaza*

Birds-of-a-Feather Lunch

W9 **Learn to Behave - Extend Your XAML with Behaviors** *Brian Noyes*	W10 **ASP.NET and Visual Studio vNext** *Robert Boedigheimer*	W11 **Windows Azure VM Role** *Eric D. Boyd*	W12 **Advanced XNA Games for Windows Phone** *Brian Peek*
W13 **Build Common Business Code for Windows Forms, WPF, Silverlight, WP7, and WinRT** *Rockford Lhotka*	W14 **MVC For WebForms Developers: Comparing and Contrasting** *Miguel Castro*	W15 **Storage Strategies in Windows Azure** *Eric D. Boyd*	W16 **ASP.NET MVC for Mobile Devices** *Rob Daigneau*
W17 **MVVM in Practice aka "Code Behind" - Free WPF** *Tiberiu Covaci*	W18 **Using jQuery to Replace the Ajax Control Toolkit** *Robert Boedigheimer*	W19 **SQL Azure Intro and What's New** *Eric D. Boyd*	W20 **LightSwitch Onramp** *Rich Dudley*

Visual Studio Live! Community Night *(Open to all VSL New York Attendees)*

Visual Studio Live! Day 3: Thursday, May 17, 2012

TH1 **Extending XAML To Overcome Pretty Much Any Limitation** *Miguel Castro*	TH2 **Advanced Debugging of ASP.NET Applications** *Tiberiu Covaci*	TH3 **SQL Server 2012: A Demo-Heavy Developer Briefing** *Roger Doherty*	TH4 **What's New in Visual Studio LightSwitch?** *Rich Dudley*
TH5 **Azure** *Peter Laudati*	TH6 **Improving Web Site Performance and Scalability While Saving Money** *Robert Boedigheimer*	TH7 **Introducing SQL Server Data Tools (codenamed "Juneau")** *Leonard Lobel*	TH8 **What's New in the .NET 4.5 BCL** *Jason Bock*
TH9 **What's New in WCF 4** *Ido Flatow*	TH10 **Fast, Faster ... Async ASP.NET** *Tiberiu Covaci*	TH11 **T-SQL Enhancements in SQL Server 2012** *Leonard Lobel*	TH12 **Static Analysis in .NET** *Jason Bock*

Lunch

TH13 **What's New in WCF 4.5** *Ido Flatow*	TH14 **ASP.NET MVC, Beyond the Basics** *Rob Daigneau*	TH15 **Microsoft's Big Play for Big Data** *Andrew Brust*	TH16 **Writing Asynchronous Code Using .NET 4.5 and C# 5.0** *Brian Peek*
TH17 **Monitoring and Troubleshooting WCF Services** *Ido Flatow*	TH18 **Creating a Data Driven Web Site Using WebMatrix and ASP.NET Razor** *Rachel Appel*	TH19 **How to Take WCF Data Services to the Next Level** *Rob Daigneau*	TH20 **Going Beyond F11: Debug Better and Faster with Visual Studio** *Brian Randell*

Conference Wrap-Up Panel with Q&A

For the complete session schedule and full session descriptions, please check the Visual Studio Live! New York web site at ***vslive.com/newyork***

**Speakers and Sessions Subject to Change.*

Introduction to Tiling in C++ AMP

Daniel Moth

Visual Studio 11 brings support for heterogeneous computing to the mainstream through C++ Accelerated Massive Parallelism (C++ AMP). I introduced C++ AMP in another article in this issue, which I consider prerequisite reading for this article. So if you haven't read it yet, please do so, starting on p. 28.

Before I introduce you to the GPU programming performance-optimization technique called "tiling," remember that in the previous article, you learned about *index<N>*, *extent<N>*, *array_view<T,N>*, *restrict(amp)* and *parallel_for_each*. You're able to use the C++ AMP API to implement your own data parallel algorithms, such as the matrix multiplication shared in the previous article and repeated here in **Figure 1**.

This article covers a prerelease technology called C++ AMP that will ship with Visual Studio 11. All information is subject to change.

This article discusses:

- The tiled_extent class
- The tiled_index class
- Matrix multiplication
- GPU memory hierarchy
- The tile_static storage class
- The tile_barrier class
- Tiled matrix multiplication

Technologies discussed:

C++ AMP, Visual Studio 11

Code download available at:

code.msdn.microsoft.com/mag201204Tiling

If you aren't familiar with matrix multiplication, here's an online reference: bit.ly/HiuP.

In this article, I'll introduce you to tiling, which is the most common optimization technique when programming GPUs. The only reason to introduce tiling in your algorithm is the extra level of performance that you can potentially achieve through data reuse and better memory-access patterns. Tiling allows you to take better advantage of the GPU memory hierarchy at a lower level than what you can do with the simple model, which you know from the introductory article.

There are two steps to tiling. First, you must explicitly tile the computation (this first step happens for you under the covers with the simple model); second, you must take advantage of tile_static memory (this second step doesn't happen for you automatically, so you have to do it explicitly yourself).

tiled_extent Class

You know that parallel_for_each accepts an extent object as its first argument. The extent describes the compute domain—that is, how many threads (size) and of what shape (dimensionality) will execute the computation. Consider the following two extent examples:

```
extent<1> e(12);   // 12 threads in a single dimension
extent<2> ee(2,6); // 12 threads in two-dimensional space
```

There's a tiled overload to parallel_for_each that accepts a tiled_extent argument. A tiled_extent describes how to break up the original extent into equally sized tiles. You can get a tiled_extent for up to a rank of only three. If you have more dimensions than that, you need to stick with the simple model or refactor your code. The total number of threads in a tile can't exceed 1,024.

The easiest way to obtain a tiled_extent object is by calling the parameterless templated tile method on extent, which returns a tiled_extent for that extent object. For the two earlier extent

Figure 1 **Matrix Multiplication, Simple Model**

```
1  void MatMul(vector<int>& vC, const vector<int>& vA,
     const vector<int>& vB, int M, int N, int W )
2  {
3    array_view<const int, 2> a(M, W, vA), b(W, N, vB);
4    array_view<int, 2> c(M, N, vC);
5    c.discard_data();
6    parallel_for_each(c.extent, [=](index<2> idx) restrict(amp)
7    {
8      int row = idx[0]; int col = idx[1];
9      int sum = 0;
10     for(int i = 0; i < b.extent[0]; i++)
11       sum += a(row, i) * b(i, col);
12     c[idx] = sum;
13   });
14   c.synchronize();
15 }
```

examples you can write something such as the following to obtain corresponding tiled_extent objects:

```
tiled_extent<6> t_e = e.tile<6>();        // e.rank==t_e.rank
tiled_extent<2,2> t_ee = ee.tile<2, 2>(); // ee.rank==t_ee.rank
```

For a pictorial representation see **Figure 2**, which shows how tiling an extent partitions the data into smaller subsets, which in C++ AMP are called tiles.

The numbers you choose to pass as template arguments must be known at compile time. They must evenly divide the global extent dimensions passed to the parallel_for_each:

- e[0]=12 is divisible by t_e.tile_extent[0]=6
- ee[0]=2 is divisible by t_ee.tile_extent[0]=2
- ee[1]=6 is divisible by t_ee.tile_extent[1]=2

For performance reasons, the smallest tile size in the least-significant dimension should be 16. Tuning the tile size can yield better or worse performance results, depending on the hardware you use. My advice is, don't try to do that—instead, pick a number that performs equally well across a broad set of hardware, starting with 16 and even multiples of it.

tiled_index Class

You know that in the parallel_for_each call you pass in a lambda with your code as the second argument. Your code receives an index object, and you can think of the index object as the thread ID. For example:

```
parallel_for_each(my_extent,[=](index<2> idx) restrict(amp){
  my_array_view[idx] = ...
});
```

When you tile the extent that you pass to the parallel_for_each, the signature of the lambda that you pass in accepts a tiled_index. A tiled_index consists of four index objects. For example, you can still get to the index object that you were expecting by using a property on the tiled_index object, as follows:

```
parallel_for_each(my_extent.tile<2,2>(),[=](tiled_index<2,2> t_idx) restrict(amp){
  my_array_view[t_idx.global] = ...
});
```

When writing tiled algorithms, you probably want to know the local index into the tile (not just the global index into the overall compute domain). You can get that index object through the local property of tiled_index. For some algorithms it's useful to know the index of the tile in relation to the other tiles in the computation, and also the global index of the tile's origin. You can access those index objects through the tile and tile_origin properties of tiled_index.

Using the two-dimensional extent of the earlier example (extent<2>(2,6).tile<2,2>()), you can see in **Figure 3** the values of the aforementioned tiled_index properties for the highlighted square.

Matrix Multiplication Revisited with Tiled parallel_for_each

In **Figure 1** you saw a matrix multiplication implementation using the simple model of C++ AMP. How would you change that algorithm so it can be explicitly tiled with what you've learned so far (using tiled_extent and tiled_index)?

The solution is shown in **Figure 4**, with the changes from the earlier listing in bold.

On line 6 I invoked the tile method on the extent, picking a tile size (16x16 in this example), and I changed the lambda to accept a tiled_index with matching tile size template arguments. In the lambda body I replaced all idx occurrences with t_idx.global (lines 8 and 12). This mechanistic conversion is what you should do first for all your C++ AMP algorithms when you decide to tile them. It's the first—but not the only—step on the journey from the simple model to the tiled model.

One thing to note as discussed earlier is that, with this change, you need to ensure the tile size picked evenly divides the global extent dimensions. My example assumes square matrices where each dimension is evenly divisible by 16. Also, it's common practice to hoist the tile size out into a static const int variable or a template argument.

In the simple matrix multiplication sample in **Figure 1**, the system tiles the computation on your behalf, behind the scenes. So it's implicitly tiled, instead of explicitly tiled, and you don't have to worry about the divisibility requirement. What the simple model can't do for you, and hence you have to do yourself, is the necessary step 2 of tiling: changing the algorithm to use tile_static memory and, typically, the usage of one or more of the other index objects. Before I delve into that, let's take a detour to understand some hardware basics.

Figure 2 **tiled_extent Is an Extent Partitioned into Tiles**

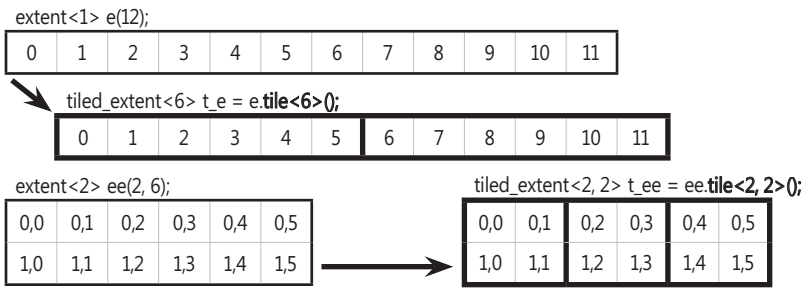


Figure 3 **tiled_index Properties Returning Index Objects**

C	col 0	col 1	col 2	col 3	col 4	col 5
row 0	130	140	150	160	170	180
row 1	290	316	342	368	394	420

For the highlighted square with the value 160:

t_idx.global	= index<2> (0,3)
t_idx.local	= index<2> (0,1)
t_idx.tile	= index<2> (0,1)
t_idx.tile_origin	= index<2> (0,2)

Figure 4 **Matrix Multiplication, Tiled, Step 1 Without Using tile_static**

```
3  array_view<const int, 2> a(M, W, vA), b(W, N, vB);
4  array_view<int, 2> c(M, N, vC);
5  c.discard_data();
6  parallel_for_each(c.extent.tile<16,16>(),
     [=](tiled_index<16,16> t_idx) restrict(amp)
7  {
8  int row = t_idx.global[0]; int col = t_idx.global[1];
9  int sum = 0;
10 for(int i = 0; i < b.extent[0]; i++)
11   sum += a(row, i) * b(i, col);
12 c[t_idx.global] = sum;
13 });
14 c.synchronize();
```

Figure 5 **Matrix Multiplication, Tiled Plus Using tile_static Memory**

```
1  void MatMul(vector<int>& vC, const vector<int>& vA,
     const vector<int>& vB, int M, int N, int W )
2  {
3    array_view<const int,2> a(M, W, vA), b(W, N, vB);
4    array_view<int,2> c(M, N, vC);
5    c.discard_data();
6
7    parallel_for_each(c.extent.tile<TS,TS>(),
       [=](tiled_index<TS,TS> t_idx) restrict(amp)
8    {
9      int row = t_idx.local[0]; int col = t_idx.local[1];
10     tile_static int locA[TS][TS], locB[TS][TS];
11     int sum = 0;
12     for (int i = 0; i < a.extent[1]; i += TS) {
13       locA[row][col] = a(t_idx.global[0], col + i);
14       locB[row][col] = b(row + i, t_idx.global[1]);
15       t_idx.barrier.wait();
16
17       for (int k = 0; k < TS; k++)
18         sum += locA[row][k] * locB[k][col];
19       t_idx.barrier.wait();
20     }
21     c[t_idx.global] = sum;
22   });
23   c.synchronize();
24 }
```

Short Background on GPU Memory Hierarchy

There's a lot of online content that discusses GPU hardware characteristics. This content differs depending on which hardware vendor it focuses on, and even what hardware family from each vendor it describes. In this section I offer a high-level, cross-hardware and rough (each hardware vendor has its own terminology and its own subtleties) picture, from a C++ AMP developer's perspective.

GPUs have global memory in which your array and array_view data resides. There are also registers for each thread, which is where your local variables typically go (unless your hardware driver has other ideas—for example, if you use too many registers for the hardware your code executes on, it will spill their contents into global memory). Accessing global memory is much slower than accessing registers—for example, it takes one GPU cycle for register access versus 1,000 cycles for a global memory access.

Additionally, near each of their processing elements, GPUs have a small scratchpad memory space (for example, 16KB to 48KB, depending on hardware). This is much faster to access than global memory; perhaps 10 cycles per access, for example. This memory area, also called the local data store, is a programmable cache. CPU caches are automatically and transparently managed for you, and hence any performance benefits are automatically granted to you. In contrast, you have to manage this GPU cache yourself by copying data into and out of it, hence, you have to opt in to the performance benefit. Some programming models call this "shared memory," others call it "local memory" and yet others call it "group shared memory." In C++ AMP this programmable cache is called "tile_static memory"—more on that later.

Next, let's map a logical model to the GPU hardware model, starting with memory lifetimes and scope.

A piece of global memory is accessible to all threads, and its lifetime exceeds that of the lifetime of a parallel_for_each computation, so a subsequent parallel_for_each invocation can operate on the same data. A register value is only accessible by one thread, and its lifetime is that of the thread. A piece of tile_static memory is shared by a subset of all the threads, which in C++ AMP is called a tile of threads, and its lifetime is that of the tile. Now you're starting to see why you need to tile your computation: Without knowing which tile a thread belongs to, you have no way of using this fast tile_static memory.

You can think of the tile_static memory as being "leased" by the tile of threads until the tile completes execution, at which point another tile takes over. So, logically speaking, threads from different tiles are isolated from each other with regard to tile_static memory and they appear to each have ownership of the tile_static memory.

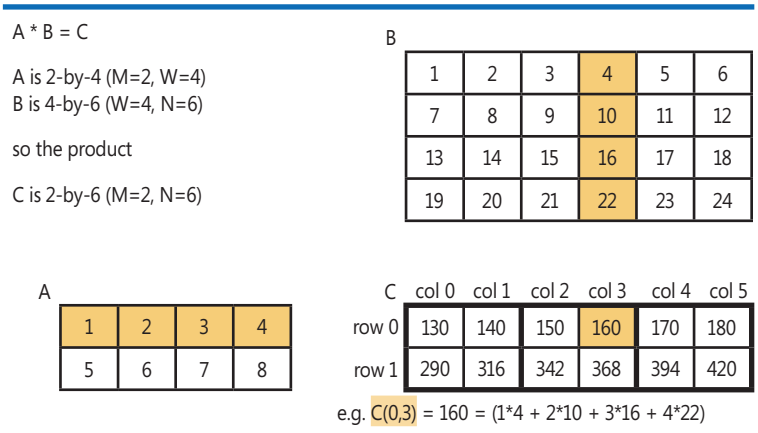
Using the New tile_static Storage Class

To access tile_static memory you use the tile_static storage class. This is the second enhancement that C++ AMP makes to the C++ language (the other being *restrict*, which I covered in my previous article).

You can only use this new storage class in restrict(amp) functions, and only if the parallel_for_each is tiled, and only for variables declared within that function block. Pointers and references can't be marked tile_static, and any implicit constructors/destructors of tile_static variables aren't called. Typically (but not always) your tile_static variables are arrays, and they are typically proportional to the tile size, for example:

```
tile_static float t[16][16]; // Access t to access tile static memory
```

Figure 6 **Matrix Multiplication Example with 12 Threads in 2-by-2 Tiles**



[Tile]	[Thread]	TS	t_idx.global	row	col	i	locA[row][col]	locB[row][col]
[0, 1]	[0, 0]	2	(0, 2)	0	0	0	1	3
[0, 1]	[0, 1]	2	(0, 3)	0	1	0	2	4
[0, 1]	[1, 0]	2	(1, 2)	1	0	0	5	9
[0, 1]	[1, 1]	2	(1, 3)	1	1	0	6	10

Figure 7 **When i=0, the Values of Other Variables Are Shown in Each Column, Where Each Row Is a Thread**

The most common way to take advantage of tile_static memory is to identify areas in global memory that you access more than once in your algorithm. You then copy those areas into tile_static memory (paying the price of global memory access only once) and then change your algorithm to use the tile_static copy of the data (accessing it very fast, repeatedly), instead of accessing the data in global memory multiple times. The programmable cache is small, so you can't copy all of your array and array_view data to tile_static variables. A tiled algorithm is typically more complex because it needs to work around that by copying only a tile-size-worth of data from global memory.

Before you look at a more real-world example that shows the technique of the preceding paragraph, let's look at a simplistic, contrived and buggy example, focused purely on the usage of tile_static where I try to sum all the elements of a matrix:

```
1  static const int TS = 2;
2  array_view<int, 2> av(2, 6, my_vector);
3  parallel_for_each(av.extent.tile<TS,TS>(),
     [=](tiled_index<TS,TS> t_idx) restrict(amp)
4  {
5    tile_static int t[TS][TS];
6    t[t_idx.local[0]][t_idx.local[1]] = av[t_idx.global];
7
8    if (t_idx.local == index<2>(0,0)) {
9      t[0][0] = t[0][0] + t[0][1] + t[1][0] + t[1][1];
10     av[t_idx.tile_origin] = t[0][0];
11   }
12 });
13 int sum = av(0,0) + av(0,2) + av(0,4); // The three tile_origins
```

The technical term for the preceding code is: horrible. Lines 9 and 13 take advantage of the fact that the tile size is 2x2=4 and that the overall size is 2x6=12 (hence three tiles), when real implementations should be written in such a way so that changing the tile size or overall extent doesn't mean changing any other code. The performance of this code is also horrible because it has a naïve algorithm and its branching is not amenable to data parallelization. There's no reuse of the data in the algorithm, so the usage of tile_static memory isn't even justified. There's also a correctness error that I'll mention later. However, as horrible as this code is, it's simple enough for someone brand new to tile_static storage to be able to understand the code—that's all that matters.

At line 6 each thread in each tile copies data to its tile_static memory from global memory. At line 9, only the first thread of each tile will sum the results for that tile into the first position of the tile_static memory for that tile. Then on line 10 that same thread (in each tile) will store the result back into global memory. Then the computation on the accelerator completes and, on the host side on line 13, the CPU thread adds up the three sums from the three tiles into the sum variable.

Did you spot a correctness error, specifically a race condition? On to the next section to learn about the final part of the tiled API, which helps us deal with that bug.

tile_barrier Class

The race condition in the previous example is between lines 6 and 9. On line 9 the thread with local index (0,0) assumes that all the values are already stored in the tile_static variable t, which is only true if all other threads in the tile have already performed line 6. Because threads generally execute independently of one another, that isn't an assumption you can make.

What you need here is a way to codify the condition that line 9 must not be executed until all threads in the tile have executed line 6. The way you express that constraint is by using a tile_barrier and calling one of its wait methods. You can't construct a tile_barrier object yourself, but you can obtain one through the barrier property from the tiled_index passed to your lambda. So you can fix the race condition by inserting the following line, after line 6:

```
7 t_idx.barrier.wait();
```

Note that you couldn't have placed this line just before line 9 and within the conditional block. A call to tile_barrier::wait must appear at a location where all threads in a tile will either reach that location or all of them will bypass it. If the barrier was allowed to be placed in such locations, then if one thread in a tile didn't execute the wait, the program would hang, waiting on that thread.

The compiler is able to flag many of these race conditions for you, and the ones it can't, the debugger can find for you if in Visual Studio 11 you turn on GPU Memory Access Exceptions under the Exceptions dialog, from the Debug | Exceptions menu.

Example: Tiled Matrix Multiplication

Do you recall the matrix multiplication example in **Figure 4**? It's now time for part 2 of the tiling exercise, where you'll also take advantage of tile_static memory and inevitably will use local indexing and the tile_barrier object.

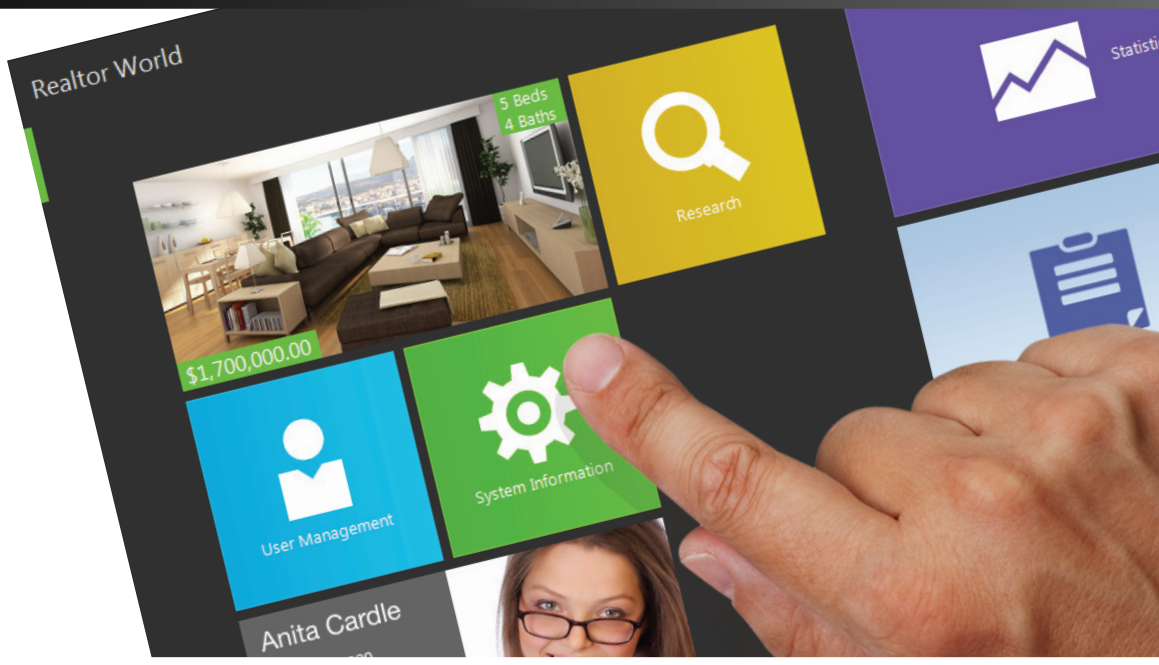
Before showing the solution, I'll remind you that on my laptop machine the simple C++ AMP matrix multiplication yields a performance improvement of more than 40 times compared to the serial CPU code for M=N=W=1024. The tiled solution that you're about to see, with TS=16 (so 16x16 = 256 threads per tile), is an additional two times faster than the simple model! That measurement includes the data transfer, so if you had transferred the data already and performed a number of computations on the data, then the kernel execution time would be much more than two times faster than the variant that doesn't use tile_static memory. So what complexity price are you paying for that kind of performance boost?

k	locA[row][k]	locB[k][col]	sum
0	1	3	3
0	1	4	4
0	5	3	15
0	5	4	20

k	locA[row][k]	locB[k][col]	sum
1	2	9	21
1	2	10	24
1	6	9	69
1	6	10	80

Figure 8 **Still i=0, the Values of Other Variables Are Shown for k=0 and k=1**

Microsoft
GOLD CERTIFIED
Partner



DXv2 is here

Touch at your fingertips.

Bring your software to life with intelligent touch-based applications. Use your existing development skills to tap into the growing demand for stunning tablet & touch-enabled apps across all platforms, including WinForms, WPF and ASP.NET. Build for today as you begin to re-imagine business applications for the Windows 8 Metro design aesthetic. DXv2 delivers the gestures, themes, and controls to put Touch within your reach, right now.

DXv2 is the next generation of tools that can take your applications to a whole new level. Your users are ready—what will you build for them? Download your free 30-day trial at **www.DevExpress.com**

www.DevExpress.com

DevExpress

[Tile][Thread]		TS	t_idx.global	row	col	i	locA[row][col]	locB[row][col]
[0, 1]	[0, 0]	2	(0, 2)	0	0	2	3	15
[0, 1]	[0, 1]	2	(0, 3)	0	1	2	4	16
[0, 1]	[1, 0]	2	(1, 2)	1	0	2	7	21
[0, 1]	[1, 1]	2	(1, 3)	1	1	2	8	22

Figure 9 **When i=2, the Values of Other Variables Are Shown in This Table**

The fully tiled matrix multiplication code is shown in **Figure 5**.

While the code in **Figure 5** works for any tile size and any total extent size (conforming to the rules described earlier), the easiest way to understand what it does is to use small numbers. Small numbers don't perform well at run time, but they do help us get a grip on what's going on. So let's assume tiles that are 2-by-2 (TS =2) and M=2, N=6 and W=4. This configuration is pictured in **Figure 6**.

To understand the code, follow just one tile, the second tile (of the three) and just one specific thread: the thread that calculates the result 160 (this is highlighted in **Figure 6**—the highlighting includes the row of A and the column of B that this thread needs access to in order to calculate the results into the cell of C).

Let's walk through the code of **Figure 5** while keeping an eye on the helpful picture of **Figure 6**.

When i=0, the Parallel Watch window in **Figure 7** shows the values of the relevant variables up until the inner loop on line 16.

As you can see, the four threads of the tile cooperated to copy data into the two tile_static arrays locA and locB from matrices A and B. Then, after they all met at the barrier on line 15, they entered the inner loop on line 17. See **Figure 8** for the relevant variable values for both k=0 and then k=1.

At this point the thread you're following calculates a partial sum of 1x4 and in the second iteration of variable k adds it to 2x10, making a total of 24 (see **Figure 8**). It then meets the other threads at the barrier on line 19. They're now ready to go into a second iteration on the outer loop. Notice that the variable i will have the value of 2, and **Figure 9** shows the relevant new values of the variables up until the barrier.

Once again, the four threads of the tile cooperated to copy data into the two tile_static arrays locA and locB from matrices A and B, moving to the right in matrix A and downward for matrix B. Then, after they all meet at the barrier on line 15, they again enter the inner loop on line 17; see **Figure 10** for the relevant variable values for both k=0 and k=1.

At this point, according to **Figure 10**, the thread you're following adds to 24 the product of 3x16 and on the second k iteration it further adds 4x22, making a grand-total sum of 160. It then meets the other threads at the barrier on line 19. The threads are now ready to go into a third iteration on the outer loop, but they realize the loop condition is no longer satisfied for variable i, so they skip to line 21. The thread you're following uses its global index to update the global memory in C.

You can see now how each matrix element was copied once from global memory by each thread and then reused by each of the other threads in the tile. As I stated earlier, the performance increase comes from copying a piece of data once from global memory into tile_static memory, and then reusing that piece of data multiple times in the code.

To help you understand the code, I chose a tile size of 4 (TS=2) plus small extents, but in real code both of those would be larger. The performance numbers I shared earlier relied on 16x16=256 threads per tile, so I was reusing a piece of data 256 times from fast memory instead of accessing it each time from the global memory. The tiled matrix multiplication implementation additionally benefits from better memory-access patterns on matrix A (while matrices B and C are accessed efficiently in all implementations), but memory coalescing is beyond the scope of this article.

That's the conclusion of the tiled matrix multiplication that takes advantage of tile_static memory. If you'd like more practice working out tiled algorithms, see the samples on the "Parallel Programming in Native Code" team blog at bit.ly/xZt05R.

The Tradeoff

In this article I introduced you to tiling, the most common optimization technique for optimizing your C++ AMP code.

You learned how to use three new C++ AMP classes (tiled_extent, tiled_index and tile_barrier), plus the tile_static storage class. You know that you can start with a simple (not explicitly tiled) implementation. Then you can modify the implementation by calling the tile function on the extent object (picking a tile size) and modifying your lambda signature to accept a tiled_index (with the same tile size template arguments).

That completed the mechanistic step 1. For the second and final step, you rewrote your algorithm to use tile_static memory with appropriate use of synchronization using a tile_barrier. That's the creative step, where for each algorithm you have to come up with a brand-new solution. The simple and tiled implementations of matrix multiplication demonstrate the complexity introduced with tiling, and also why its usage can result in performance gains. The tradeoff is yours to opt in to, or not.

It's worth mentioning that this kind of cache-aware programming can result in speed gains for your CPU code, too, because you'd be helping the automatic transparent cache-management system do a better job. Also, your tiled code becomes more amenable to vectorization by smart compilers.

Beyond my two articles, there's a wealth of C++ AMP content (documentation, tips, patterns, links to videos and samples) on the Parallel Programming team blog (bit.ly/bWfC5Y) that I strongly encourage you to read. ■

k	locA[row][k]	locB[k][col]	sum
0	3	15	66
0	3	16	72
0	7	15	174
0	7	16	192

k	locA[row][k]	locB[k][col]	sum
1	4	21	150
1	4	22	160
1	8	21	342
1	8	22	368

Figure 10 **Still i=2, the Values of Other Variables Are Shown for k=0 and k=1**

DANIEL MOTH *is a principal program manager in the Microsoft Developer Division. He can be reached via danielmoth.com/Blog.*

THANKS to the following technical experts for reviewing this article: *Steve Deitz, Yossi Levanoni, Robin Reynolds-Haertle, Stephen Toub and Weirong Zhu*

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

US Sales: 1.888.277.6734
sales@aspose.com
EU Sales: +44 (0)800 098 8425
sales.europe@aspose.com

ASPOSE
Your File Format Experts

Lowering the Barriers to Code Generation with T4

Peter Vogel

The Microsoft .NET Framework makes extensive use of code generation both at design time (when dragging a control onto a design surface generates code) and at run time (when LINQ generates the SQL statements that retrieve data). Code generation obviously makes developers more productive by reducing the amount of code a developer has to write, but it can be especially useful when more-or-less identical code is used in many solutions. As you implement this similar (but not identical) code in a new application, it's all too easy to introduce new errors.

Even though developers can take advantage of all of the code-generation tools the .NET Framework uses in creating applications, very few make extensive use of code generation in their day-to-day development practice. There are a number of reasons for this: they worry that incorporating code generation requires a new toolset they don't have the time to learn; they lack experience in recognizing problems that code generation will solve and in designing solutions that integrate generated code with "handwritten" code; and they understand that some code-generation solutions might require more time to maintain (or even to use) than will be saved by applying the solution.

This article discusses:

- Recognizing when a code-generation solution would be useful
- Creating a code-generation template
- Managing the code-generation process
- Using the code-generation package
- Code-generation tools and resources

Technologies discussed:

Microsoft .NET Framework, Visual Studio Text Template Transformation Toolkit, Visual Studio 2010

Code generation obviously makes developers more productive.

The Microsoft Text Template Transformation Toolkit (T4) addresses many of these issues, providing a simple way to implement code-generation solutions that leverage tools and techniques developers are already comfortable with. T4 does this by recognizing that a typical solution involves two types of code: boilerplate code that doesn't change from one code generation to another, and dynamic code that does change. T4 simplifies code generation by allowing developers to simply type the boilerplate code portion of a solution into a file. In T4, the dynamic code (typically a small part of a code-generation solution) is generated using a set of tags that look very similar to the tags an ASP.NET MVC developer would use in creating a View, or that an ASP.NET developer would use to embed server-side code in an .aspx file.

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

YOUR MAP TO THE .NET DEVELOPMENT PLATFORM

Brooklyn, NY | **May 14-17** | NY Marriott at the Brooklyn Bridge

BIG CODE IN THE BIG APPLE

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

Check out the hot track topics that will make YOU a more valuable part of your company's development team:

- **Windows 8 / WinRT**
- **Silverlight / WPF**
- **Web**
- **Visual Studio 2010+ / .NET 4.0**
- **Cloud Computing**
- **Data Management**
- **HTML5**
- **Windows Phone 7**
- **Cross Platform Mobile**

vslive.com/newyork

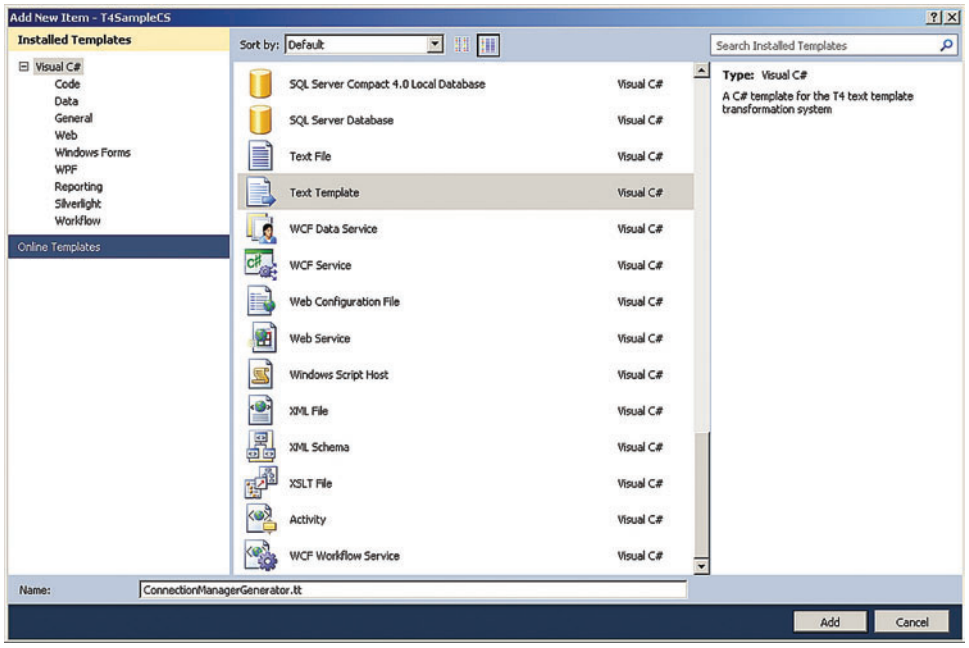


Figure 1 **Adding a T4 Template**

Using a T4-based solution leverages the skills you already have by letting you specify the inputs for generating code in whatever programming language you're already using. T4 doesn't generate code-neutral solutions (the code generated by a T4 solution is always in some specific programming language), but most developers don't need code-neutral solutions.

Defining Code-Generation Solutions

While T4 makes it easier to create code-generation solutions, it doesn't address the problems developers have in recognizing when a code-generation solution would be useful and in actually designing a solution. Problems that code generation solves generally share three characteristics:

1. The generated code goes into a separate file from the developer's code. This ensures the code-generation process won't interfere with the developer's code. Typically, this means the generated code goes into a new class that the developer will use from his "handwritten" code. It might make sense to generate a partial class that the developer can not only call but extend—but the developer's code and the generated code are still kept in separate files.
2. The generated code is repetitive: The code in the solution is a template that can be repeated many times, often with small variations. This ensures the code generation is simpler and easier to maintain than the equivalent handwritten code.
3. Compared to the handwritten solution, the generated solution requires far fewer inputs (ideally, no inputs at all—the code-generation solution determines what needs to be done from the environment). If the number of inputs is large or hard to determine, then developers might regard the handwritten solution as simpler than the generated solution.

Given these characteristics, three types of scenarios are worth investigating for code generation. Developers tend to focus on the first scenario ("the ultimate scenario"), in which just a few inputs are used to generate a lot of code that's used frequently (think of the Entity Framework, for instance). In fact, the most fruitful code-generation opportunities fall into two other scenarios.

The second scenario is the most obvious one: When a great deal of code needs to be generated. In that case, avoiding writing multiple lines of repetitive code is clearly beneficial. While most code-generation solutions are used in multiple apps, solutions in this category can be worthwhile even if used in only a single application. Rather than write generalized code with many If statements to handle each situation—each If statement doubling the logical complexity of the code—a code-generation solution can generate the specific code required for each set of conditions. Instead of hand coding many classes that share an interface, code generation can be used to create the individual classes (assuming the classes share a common structure).

> Very few programmers make extensive use of code generation in their day-to-day development practice.

But the third type of scenario is the most common: When a few inputs will generate a little bit of code to be used in many applications. In this scenario, the amount of repeated code in any particular application is small but the activity the code supports is so common that the solution ends up generating a great deal of code—just not in any one application.

For example, here's some typical code that ADO.NET developers write all the time:

```
string conString =
  System.Configuration.ConfigurationManager.
    ConnectionStrings["Northwind"].ConnectionString;
```

Though this is a trivial amount of code, it's code that's repeated—with just the name of the connection string changing—in application after application. Moreover, in the absence of any IntelliSense support for the connection string name, the code is error-prone: it's open to "spelling counts" errors that will be discovered only at run time (probably when someone who has input to your appraisal is looking over your shoulder). Another example is implementing INotifyPropertyChanged, which leaves the developer open to "spelling counts" errors on each property.

This code for retrieving a connection string named Northwind would be more useful if some code-generation solution existed for creating a ConnectionManager class for each connection string, like this:

```
string conString = ConnectionManager.Northwind;
```

Once you recognize an opportunity for a code-generation solution, the next step is to write out a sample of the code that the solution would generate. In this case, the ConnectionManager class might look like this:

```
public partial class ConnectManager
{
  public static string Northwind
    {
      get
        {
          return System.Configuration.ConfigurationManager.
            ConnectionStrings["Northwind"].ConnectionString;
        }
    }
}
```

This code meets the criteria for a code-generation solution: it's repetitive (the property code repeats for each connection string) with only small changes (the name of the connection string and the property name) and the number of inputs is small (just the names of the connection strings).

Your First Code-Generation Template

A T4 solution can consist of a "code-generation package": a file where the developer inserts the inputs to the code-generation process and a template file that generates code from those inputs. Both files are T4 template files and are created using the same programming tools you use to write your applications. This design allows you to separate your code-generation template from the file the developer uses to provide the inputs to the process.

To begin creating your code-generation solution, you must add the T4 template file that will generate your code to an application where you can test it—preferably, an application similar to the ones you expect your solution to be used in. To add the T4 template file, in the Add New Item dialog in Visual Studio, shown in **Figure 1**, add a Text Template specifying a name appropriate for your code-generation template (for example, ConnectionManagerGenerator). If your version of Visual Studio doesn't have the Text Template option, add a new Text File (also found in the General section), giving the file the extension ".tt" to trigger T4 processing. If you do add a Text file you'll get a warning message that you can safely ignore.

If you examine the properties for your new template file, you'll find that its Custom Tool property has been set to TextTemplatingFileGenerator. This custom tool is run automatically by Visual Studio and is the host that manages the code-generation process. In T4, the contents of the template file are passed to the code-generation host, which puts the resulting generated code in the template file's nested child file.

If you've added a Text file to your project, your template file will be empty; if you were able to add a Template file, it will contain two T4 directives, marked with <#@...#> delimiters (if you added a Text file, you'll need to add these directives). These directives specify the language that the template will be written in (*not* the language for the generated code) and the extension for the child file. In this example, the two directives set the programming language for the template to Visual Basic and the file extension for the child file containing the generated code to .generated.cs:

```
<#@ template language="VB" #>
<#@ output extension=".generated.cs" #>
```

> Eventually, you'll need to add control code that will dynamically generate the output code.

To create the traditional "Hello, World" application, just add the code to the template file (notice that while the language the template is being written in is Visual Basic, the template is generating C# code):

```
public class HelloWorld
{
  public static string HelloWorld(string value)
  {
    return "Hello, " + value;
  }
}
```

This sample uses only boilerplate code. In T4, boilerplate code is copied straight from the template into the code file. In Visual Studio 2010, that should happen when you switch away from the template file or save it. You can also trigger code generation either by right-clicking on the template file in Solution Explorer and selecting Run Custom Tool from its context menu or by clicking the Transform All Templates button at the top of Solution Explorer.

After triggering generation, if you open the template's code file (which will now have the extension specified in the template's output directive), you'll find it contains the code specified in your template. Visual Studio will also have done a background compile

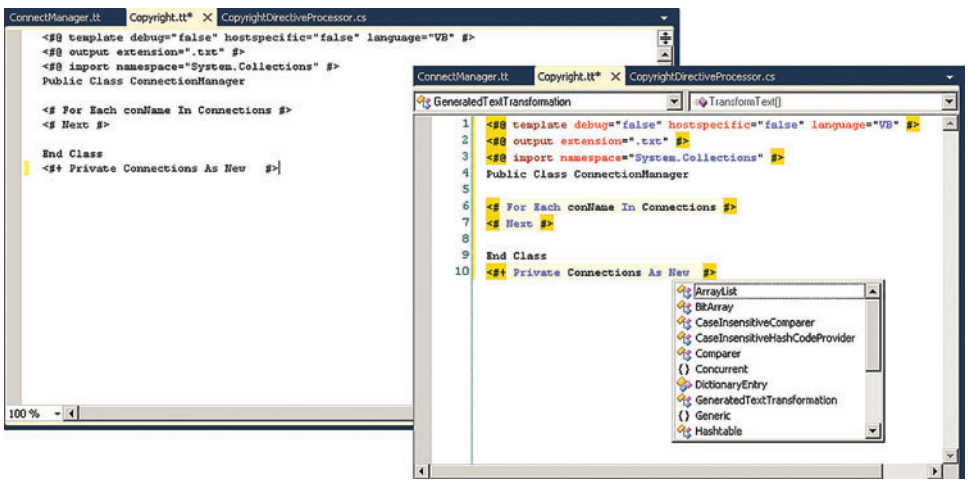


Figure 2 **The Default Editor for T4 Template Files (Left) Isn't Much Better than NotePad—Adding the Tangible Editor (Right) Gives You the Kind of Features You Expect in Visual Studio**

EXTREME SCALABILITY

As a software architect, you know that delivering extreme scalability is the key to keeping up with the demands of your users. You can't let bottlenecks to data access or analysis get in your way. You need **ScaleOut StateServer's** distributed data grid to maximize your application's performance.

Its powerful architecture automatically scales in-memory data storage across multiple servers to combine blazing speed with industry-leading ease of use. Unique management tools and "map/reduce" parallel analysis give you everything you need to take your application's scalability to the next level.

Download your **FREE** trial copy of ScaleOut StateServer® today!

SCALEOUT SOFTWARE

Distributed Data Grids for the Enterprise

www.scaleoutsoftware.com/trial | 503.643.3422

of your new code, so you'll find you can use the generated code from the rest of your application.

Generating Code

Boilerplate code isn't sufficient, however. The ConnectionManager solution must dynamically generate a property for each connection string the application requires. To generate that code, you must add control code to manage the code-generation process and some variables that will hold the inputs from the developer using your code-generation solution.

The ConnectionManager uses an ArrayList (which I've called Connections) from the System.Collections namespace to hold the list of connection strings that form the input to the code-generation process. To import that namespace for use by code within your template, you use the T4 Import directive:

```
<#@ Import Namespace="System.Collections" #>
```

Now you can add any static code that begins your generated class. Because I'm generating C# code, the initial code for the ConnectionManager looks like this:

```
public partial class ConnectionManager
{
```

I must now add the control code that will dynamically generate the output code. Code that controls generation (code that is to be executed, rather than copied to the child file) must be enclosed in the <#...#> delimiters. In this example, to make it easy to distinguish between the control code and the code being generated, I've written the control code in Visual Basic (this is not a requirement of the code-generation process). The control code for the ConnectionManager solution loops through the Connections collection for each connection string:

```
<#
  For Each conName As String in Connections
#>
```

In addition to any control code in your template, you'll also need to include any expressions whose values are to be incorporated into your dynamically generated code. In the ConnectionManager solution, the name of the connection string has to be incorporated into the property's declaration and into the parameter passed to the ConnectionStrings collection. To evaluate an expression and have its value inserted into the boilerplate code, the expression must be enclosed in the <#=…#> delimiters. This example dynamically inserts the value from the conName variable into two places in the static code inside the For Each loop:

```
public static string <#= conName #>
{
  get
  {
    return System.Configuration.ConfigurationManager.
      ConnectionStrings["<#= conName #>"].ConnectionString;
  }
}
<#
  Next
#>
}
```

All that's left is to define the ArrayList that will hold the list of connection string names. For this, I'll use a T4 class feature. T4 class features are typically used to define helper functions but can also be used to define fields or any other class-level items that will be used during the code-generation process. Class features must appear at the end of a template, as this one does:

```
<#+
  Dim Connections As New ArrayList()
#>
```

This T4 template forms the first part of the ConnectionManager solution—the code-generation template. You now need to create the second part of the solution: The input file that the developer will use to provide the inputs to the code-generation process.

Using the Code-Generation Package

To provide a place for the developer to enter the inputs to the code-generation process, you add a second T4 template to the application in which you're testing your code-generation solution. This template must have an Include directive that copies your code-generation template into this template. Because I named my code-generation template file ConnectionManagerGenerator, the input template file for the ConnectionManager solution looks like this:

```
<#@ template language="VB" #>
<#@ output extension=".generated.cs" #>
<#@ Import Namespace="System.Collections" #>
<#

#>
<#@ Include file="ConnectionManagerGenerator.tt" #>
```

When code generation is performed, the host process actually assembles an intermediary .NET program from the directives, control code and static code specified in your templates, and then executes the resulting program. It's the output from that intermediary program that's poured into your template's child file. The result of using the Include directive is to merge your code-generation template (with its declaration of the Connections ArrayList) with the contents of this file to create that intermediary program. All the developer using your solution has to do is add the code to this template that will set the variables used by your code-generation template. This process allows developers to specify the inputs to the code generation using the programming language they're used to.

For the ConnectionManager solution, the developer needs to add the name of the connection strings listed in the application's app.config or web.config file to the Connections ArrayList. Because these settings are part of the control code that needs to be executed, that code must be enclosed within the <#...#> delimiters. The developer's code must also precede the Include directive so that the variables are set before your code executes.

To generate a ConnectionManager for two connection strings called Northwind and PHVIS, the developer would add this code to the input template before the Include directive:

```
<#
  Me.connections.Add("Northwind")
  Me.connections.Add("PHVIS")
#>
<#@ Include file="ConnectionManagerGenerator.tt" #>
```

You now have a code-generation package that consists of the code-generation template file and the input template file. Developers using your solution must copy both files into their application, set the variables in the input file, and close or save the input file to generate the solution code. An enterprising code-generation developer could set up the code-generation package as a Visual Studio item template that includes both template files. While not appropriate to the ConnectionManager solution, if a developer needs to generate another set of code based on different

5 YEARS OF EXCELLENCE

XCEED
DataGrid
for WPF

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

IBM®
U2 SystemBuilder™

"IBM U2 researched existing third-party solutions and identified Xceed DataGrid for WPF as the most suitable tool. The datagrid's performance and solid foundation take true advantage of WPF and provide the extensibility required for IBM U2 customers to take their applications to the next level in presentation design and styling."

Vincent Smith
U2 Tools Product Manager at IBM

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 feature we needed. Working with Xceed has been a pleasure."

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

XCEED Professional Themes for WPF

Theme your entire app in minutes. Flawless styles for all official WPF controls.

XCEED DataGrid for Silverlight

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

XCEED Ultimate ListBox for WPF

NEW

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

XCEED Ultimate ListBox for Silverlight

NEW

Fast and fluid, with ground-breaking streaming technology.

Visual Studio PARTNER

XCEED MULTI-TALENTED COMPONENTS

inputs, he would just need to make a second copy of the input file to hold the second set of inputs.

There's one wrinkle in this solution's structure: Any application that uses this solution will have both the input template and the code-generation template. In the ConnectionManager solution, if both templates generate code that Visual Studio compiles, the two resulting code files will both define a class called ConnectionManager and the application won't compile. There are a number of ways of preventing this, but the simplest way is to alter your code-generation template so that its generated code file has an extension that Visual Studio won't recognize. Changing the output directive in the code-generation template file does the trick:

```
<#@ output extension=".ttinclude" #>
```

Visual Studio essentially treats T4 templates as text files.

Code-Generation Tools and Resources

Besides the MSDN Library pages, your best source for information on using T4 is Oleg Sych's blog at olegsych.com—I've certainly come to depend on his insights (and tools) in developing my own code-generation solutions. His T4 Toolbox includes several templates for developing T4 solutions (including a template for generating multiple output files from a single T4 template and other tools for managing the code-generation process). Sych's toolkit also includes packages for several code-generation scenarios.

Visual Studio essentially treats T4 templates as text files—which means you don't get IntelliSense support or highlighting or, really, anything that developers expect from an editor. In Visual Studio Extension Manager, you'll find several tools that will enhance your T4 development experience. Both Visual T4 from Clarius Consulting (bit.ly/maZFLm) and T4 Editor from Devart (bit.ly/wEVEVa) will give you many of the features you take for granted in an editor. Alternatively, you can get the T4 Editor (either the free or PRO EDITION) from Tangible Engineering (see **Figure 2**) at bit.ly/16jvGY, which includes a visual designer you can use to create code-generation packages from Unified Modeling Language (UML)-like diagrams.

As with any other code-based solution, it's unlikely your solution will work the first time. Compile errors in your template control code are reported in the Errors list after you select Run Custom Tool from a template file's context menu. However, even if your control code compiles, you might find that the template's child file is empty except for the word ErrorGeneratingOutput. This indicates that the control code in your code-generation package is generating an error when it executes. Unless your error is obvious, you're going to need to debug that control code.

To debug your generation package, you must first set the debug attribute on the template directive to True, like this:

```
<#@ template language="VB" debug="True"#>
```

You can now set a break point in your control code and have Visual Studio respect it. Then, the most reliable way to debug your application is to start a second version of Visual Studio and, from the Debug menu, select Attach to Process. In the resulting dialog, you select the other running copy of devenv.exe and click the Attach button. You can now return to your original copy of Visual Studio and use Run Custom Tool from your input file's context menu to start executing your code.

If that process doesn't work, you can trigger debugging by inserting this line into your template's control code:

```
System.Diagnostics.Debugger.Break()
```

With Visual Studio 2010 on Windows 7, you should add this line before your call to the Break method:

```
System.Diagnostics.Debugger.Launch()
```

When you execute your template code, this line brings up a dialog box that lets you restart Visual Studio or debug it. Selecting the debug option will start a second copy of Visual Studio already attached to the process running your template code. Your initial instance of Visual Studio will be disabled while you're debugging your code. Unfortunately, when your debugging session is over, Visual Studio will stay in that disabled mode. To prevent this, you'll need to alter one of the Visual Studio settings in the Windows Registry (see Sych's post on debugging T4 at bit.ly/aXJwPx for the details). You'll also need to remember to delete this statement once you've fixed your problem.

Of course, this solution still counts on the developer entering the names of the connection strings correctly into the input file. A better solution would have ConnectionManager include code that reads the connection strings from the application's config file, eliminating the need for the developer to enter any inputs. Unfortunately, because code is being generated at design time rather than at run time, you can't use the ConfigurationManager to read the config file and will need to use the System.XML classes to process the config file. You'll also need to add an assembly directive to pick up those classes, as I did earlier to get ArrayList out of System.Collections. You can also add references to your own custom libraries by setting the assembly directive's name attribute to the full path to your DLL:

```
<#@ assembly name="C:\PHVIS\GenerationUtilities.dll" #>
```

As with any other code-based solution, it's unlikely your solution will work the first time.

These are the essentials for adding code generation to your toolkit and increasing your productivity—along with your code's quality and reliability. T4 makes it easy for you to get started by letting you create code-generation solutions using a familiar toolset. The biggest problem you'll face in using T4 is learning to recognize opportunities for applying these tools. ■

PETER VOGEL *is a principal in PH&V Information Services. His last book was "Practical Code Generation in .NET" (Addison-Wesley Professional, 2010). PH&V Information Services specializes in facilitating the design of service-based architectures and in integrating .NET technologies into those architectures. In addition to his consulting practice, Vogel wrote Learning Tree International's SOA design course, which is taught worldwide.*

THANKS to the following technical expert for reviewing this article: *Gareth Jones*

Integrating Windows Workflow Foundation with the OpenXML SDK

Rick Spiewak

Business processes that involve a workflow often require that corresponding documents be created or processed. This can occur, for example, when an application (for a loan, an insurance policy, redemption of shares and so forth) has been approved or rejected during the workflow process. This might be driven by a program (automatically) or by an underwriter (as a manual step). In that case, a letter might need to be written, or a spreadsheet showing balances produced.

In the June 2008 issue of *MSDN Magazine*, I showed how to accomplish this using the Microsoft Office application object models (msdn.microsoft.com/magazine/cc534981). Using that article as a basis, this time I'll show you how to integrate Microsoft Office-compatible documents with Windows Workflow Foundation (WF) without having to interact directly with Office applications. This is achieved by using the OpenXML SDK 2.0 to manipulate word-processing and spreadsheet document types. The corresponding Office applications are, of course, Word and Excel. While each of these has its own OpenXML document model, there are enough similarities to allow the use of a set of interface classes that hide most of the differences for the purpose of workflow integration.

Because WF is included in the Microsoft .NET Framework 4 Client Profile, any .NET Framework 4 installation will include the WF library. And because its use has been greatly simplified in the .NET Framework 4 as compared with the .NET Framework 3.0, any application that requires even basic workflow functions should consider using this in place of custom code. This applies even to cases where the built-in activities need to be supplemented by custom activities.

I'll show how a custom activity can provide data entry based on an Office document that serves as a prototype. The data entry activity, in turn, passes data to activities for each document type, and the activities use these data fields for filling in target Office documents. I used Visual Studio 2010 to develop classes to support operations such as enumerating named fields, extracting their contents and filling in documents from prototypes. These classes all use the OpenXML SDK rather than directly manipulating Office application object models. I created workflow activities to support data entry and filling in the

This article discusses:

- Interfacing with Word and Excel documents via the OpenXML SDK
- Building custom Activities
- Passing data in to Workflow Activities
- Extracting Workflow data and using it in Office documents

Technologies discussed:

Windows Workflow Foundation 4.0, Microsoft .NET Framework 4, Windows Presentation Foundation, OpenXML SDK 2.0, Microsoft Office, Visual Studio 2010, Visual Basic

Code download available at:

code.msdn.microsoft.com/mag201204OpenXML

Figure 1 **Loading and Saving an OpenXML Document**

```
Public Sub LoadDocumentToMemoryStream(ByVal documentPath As String)
  Dim Bytes() As Byte = File.ReadAllBytes(documentPath)
  DocumentStream = New MemoryStream()
  DocumentStream.Write(Bytes, 0, Bytes.Length)
End Sub

Public Sub SaveDocument()
  Dim Directory As String = Path.GetDirectoryName(Me.SaveAs)
  Using OutputStream As New FileStream(Me.SaveAs, FileMode.Create)
    DocumentStream.WriteTo(OutputStream)
    OutputStream.Close()
    DocumentStream.Close()
  End Using
End Sub
```

word processing and spreadsheet document types. Finished documents are displayed by simply invoking the default application for the document type. I wrote the code using Visual Basic.

Overall Design

Any design for integrating workflow and Office has three general requirements: getting data into the workflow; processing the data to create or update Office documents; storing or passing on the output documents. To support these needs, I created a set of classes that provide uniform interfaces to the underlying Office document formats using the OpenXML SDK. These classes provide methods to:

- Get the names of the potential target fields in the documents. I'll use the generic term "field" to describe these. In the case of Word, these are bookmarks; Excel uses named ranges. In the most general case, both bookmarks and named ranges can be quite complex, but I'll use the simple case of a single location for bookmarks and single cells for named ranges. Bookmarks always contain text, whereas spreadsheet cells can contain text, numbers or dates.
- Fill in target fields in a document by accepting input data and matching the data to the document.

To implement activities to fill in the Office documents, I used the WF 4 CodeActivity model. This model is greatly simplified over WF 3.0 and provides a much clearer implementation. For example, it's no longer necessary to explicitly declare dependency properties.

The Supporting Cast

Behind the workflow stands a set of classes designed to support the functions of the OpenXML SDK. The classes perform the key functions of loading the prototype document into a memory stream, finding and filling in the fields (bookmarks or named ranges), and saving the resulting output document. Common functions are collected into a base class, including loading and saving the memory stream. I've omitted error checking here for clarity, but it's included in the accompanying code download. **Figure 1** shows how to load and save an OpenXML Document.

Getting Data into the Workflow

One of the first challenges I faced in integrating Office documents into workflows was how to pass data destined for fields in these documents. The standard workflow structure relies on advance knowledge of the names of variables associated with activities. Variables can be defined with different scopes to provide visibility into the workflow, and into other activities. I decided that applying this model directly was too rigid, as it required tying the overall workflow design too tightly to the document fields. In the case of Office integration, the workflow activity is acting as a proxy for an Office document. It isn't realistic to try to predetermine the names of the fields in the document, as this would require matching a custom activity to a particular document.

If you look at the way arguments are passed to activities, you'll find they're passed as a Dictionary(Of String, Object). To fill in fields in an Office document, you need two pieces of information: the name of the field and the value to insert. I've developed applications using workflow products and Office before, and the general strategy I adopted worked well: I enumerate the named fields in the document and match them to input parameters by name. If the document fields match the pattern in the basic input parameter dictionary (String, Object), they can be passed in directly. However, this approach couples the workflow too tightly to the document.

Rather than naming variables to correspond with fields in the document, I decided to use a generic Dictionary(Of String, String) to convey these field names. I named this parameter Fields, and used it in each of the activities. Each entry in such a dictionary is of type KeyValuePair(Of String, String). The key is used to match the field name; the value is used to fill in the field contents. This Fields dictionary is then one of the parameters inside the workflow.

You can start the workflow with just a few lines of code in a simple Windows Presentation Foundation (WPF) window, and even fewer when added to an existing application:

```
Imports OfficeWorkflow
Imports System.Activities
Class MainWindow
  Public Sub New()
    InitializeComponent()
    WorkflowInvoker.Invoke(New Workflow2)
    MessageBox.Show("Workflow Completed")
    Me.Close()
  End Sub
End Class
```

I wanted the activities to be generally useful, and to have more than one strategy available for providing the input document. To allow for this, the activities have a common parameter named

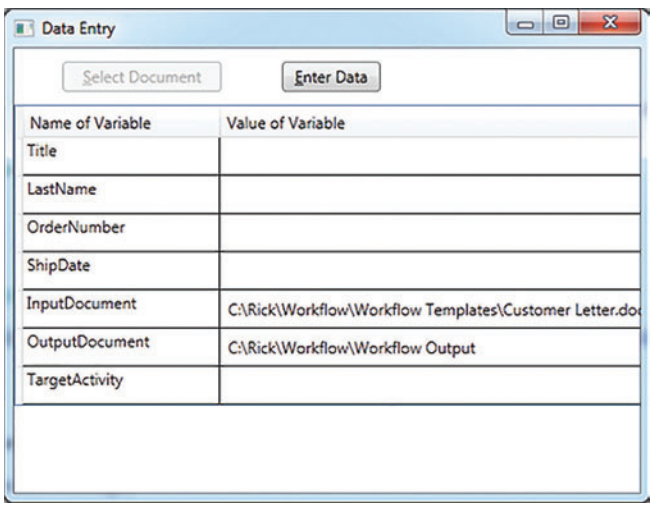


Figure 2 **The Data Entry Activity Interface**

Figure 3 **Filling in a Word-Processing Document in the Activity**

```
Protected Overrides Sub Execute(ByVal context As CodeActivityContext)
  InputFields = Fields.Get(Of Dictionary(Of String, String))(context)

  InputDocumentName = InputDocument.Get(Of String)(context)
  If String.IsNullOrEmpty(InputDocumentName) Then InputDocumentName = _
    InputFields("InputDocument")
  OutputDocumentName = OutputDocument.Get(Of String)(context)
  If String.IsNullOrEmpty(OutputDocumentName) Then OutputDocumentName = _
    InputFields("OutputDocument")

  ' Test to see if this is the right activity to process the input document
  InputFields.TryGetValue(("TargetActivity"), TargetActivityName)
  ' If there is no explicit target, see if the document is the right type
  If String.IsNullOrEmpty(TargetActivityName) Then
    If Not (InputDocumentName.EndsWith(".docx") _
    Or InputDocumentName.EndsWith(".docm")) _
    Then Exit Sub
    'If this is the Target Activity, fix the document name as needed
  ElseIf TargetActivityName = Me.DisplayName Then
    If Not (InputDocumentName.EndsWith(".docx") _
    Or InputDocumentName.EndsWith(".docm")) _
      Then InputDocumentName &= ".docx"
    End If
  Else
    Exit Sub
  End If

  ' This is the target activity, or there is no explicit target and
  ' the input document is a Word document
  Dim InputWordInterface = New WordInterface(InputDocumentName, InputFields)
  If Not (OutputDocumentName.EndsWith(".docx") _
  Or OutputDocumentName.EndsWith(".docm")) _
    Then OutputDocumentName &= ".docx"
  InputWordInterface.SaveAs = OutputDocumentName
  Dim Result As Boolean = InputWordInterface.FillInDocument()

  ' Display the resulting document
  System.Diagnostics.Process.Start(OutputDocumentName)
End Sub
```

InputDocument. These can be attached to variables that, in turn, are connected to outputs of other activities as the needs of the workflow dictate. The parameter contains the path to an input document used as a prototype. However, the code also provides for using a Field parameter whose name is InputDocument if it contains a path to a document type suitable for the target Office application. An additional parameter allows the target activity to be named in an input field named TargetActivity. This allows for multiple activities in a sequence; for example, to evaluate the fields in the original input document for applicability.

The standard workflow structure relies on advance knowledge of the names of variables associated with activities.

Any real workflow will have a source of input data. For demonstration purposes, I used a DataEntry activity, which can draw its input (the fields and default values) from any of the supported Office document types. This activity opens a dialog box containing a DataGrid and buttons for selecting a document and saving the data fields. After the user selects a document as a prototype, the DataGrid is filled in with the available fields from the document, plus the InputDocument, OutputDocument and TargetActivity fields, as shown in **Figure 2**. (Incidentally, Julie Lerman's April 2011 Data Points column, "Composing WPF DataGrid Column Templates for a Better User Experience," which you'll find at msdn.microsoft.com/magazine/gg983481, has an important tip on the use of FocusManager to enable single-click editing in a grid such as the one in **Figure 2**.)

Figure 4 **Using the WordInterface Class to Fill in a Word Document**

```
Public Overrides Function FillInDocument() As Boolean
  Dim Status As Boolean = False
  ' Assign a reference to the existing document body.
  Dim DocumentBody As Body = WordDocument.MainDocumentPart.Document.Body
  GetBuiltInDocumentProperties()
  Dim BookMarks As Dictionary(Of String, String) = Me.GetFieldNames(DocumentBody)

  ' Determine dictionary variables to use -
    based on bookmarks in the document matching Fields entries
  If BookMarks.Count > 0 Then
    For Each item As KeyValuePair(Of String, String) In BookMarks
      Dim BookMarkName As String = item.Key
      If Me.Fields.ContainsKey(BookMarkName) Then
        SetBookMarkValueByElement(DocumentBody, BookMarkName,
Fields(BookMarkName))
      Else
        ' Handle special case(s)
        Select Case item.Key
          Case "FullName"
            SetBookMarkValueByElement(DocumentBody, _
            BookMarkName, GetFullName(Fields))
        End Select
      End If
    Next
    Status = True
  Else
    Status = False
  End If

  If String.IsNullOrEmpty(Me.SaveAs) Then Return Status
  Me.SaveDocument(WordDocument)
  Return Status
End Function
```

Processing the Data into the Documents

As I noted earlier, each of the Office document types has its own way of providing a collection of named fields. Each of the activities is written to support a particular document type, but the activities that support the Office document types all follow the same pattern. If the InputDocument property is provided, it's used as the path to the document. If the InputDocument property is null, the activity looks in the Fields property for an InputDocument value that, if found, is examined to see whether it contains a path with a suffix matching the document type the activity handles. The activity also attempts to find a document by appending a suitable suffix. If these conditions are met, the InputDocument property is set to this value, and processing proceeds.

Each matching entry from the Fields collection is used to fill in the corresponding field in the output document. This is either passed in as the corresponding workflow variable (OutputDocument), or is found in the Fields collection as the OutputDocument KeyValuePair entry. In each case, if the output document has no suffix, an appropriate default suffix is appended. This allows the same value to potentially be used to create different document types, or even multiple documents of different types.

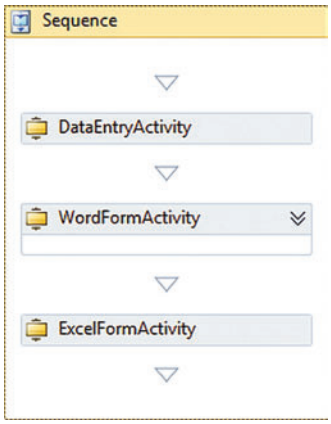


Figure 5 **Simple Workflow with Office Integration**

The output document will be stored at the path specified. In most real-world workflow environments, this would be a network share or a SharePoint folder. For simplicity, I used a local path in the code. Also, the code for the Word and Excel activities checks the input document for the corresponding type. In the demonstration workflow, the input document defaults to the prototype selected as the basis for the Fields. The user can change this, so that Fields derived from a Word document can be used to define inputs for an Excel document, or vice versa. **Figure 3** shows the code for filling in a Word document.

In **Figure 3**, note in particular the following line:

```
Dim InputWordInterface = _
  New WordInterface(InputDocumentName, InputFields))
```

This is where the WordInterface class instance is constructed. It's passed the path to the document to use as a prototype, along with the field data. These are simply stored in the corresponding properties for use by the class's methods.

The WordInterface class provides the function that fills in the target document. The input document is used as a prototype, from which an in-memory copy of the underlying OpenXML document is created. This is an important step—the in-memory copy is the one that's filled in and then saved as the output file. Creation of the in-memory copy is the same for each document type and is handled in the base OfficeInterface class. However, saving the output file is more specific to each type. **Figure 4** shows how the Word document is filled in by the WordInterface class.

I added a special case Field name called FullName. If the document contains a field with this name, I concatenate input fields called Title, FirstName and LastName to fill it in. The logic for this is in a function called GetFullName. Because all the Office document types have similar needs, this function is in the base OfficeInterface class along with some other common properties. I've used a Select Case statement to make this an extensibility point. You could, for example, add a FullAddress field that concatenates input fields called Address1, Address2, City, State, ZipCode and Country.

Storing the Output Documents

Each of the activity classes has an OutputDocument property, which can be set by several means. Within the designer, a property can be bound to a workflow-level parameter or to a constant value. At run time, each of the activities will look in its OutputDocument property for the path to save its document. If this isn't set, it will look within its Fields collection for a key named OutputDocument. If this value ends with an appropriate suffix, it's used directly. If it doesn't have an appropriate suffix, one is added. The activity then saves the output document. This allows maximum flexibility in deciding where to put the path to the output document. Because the suffix is omitted, the same value can be used by either of the activity types. Here's how a Word document is saved, first ensuring that the memory stream is updated and then using the method in the base class:

```
Public Sub SaveDocument(ByVal document As WordprocessingDocument)
  document.MainDocumentPart.Document.Save()
  MyBase.SaveDocument()
End Sub
```

Sample Workflows

Figure 5 shows the simple workflow I'll use to show how the integration works. A second example uses a flowchart, shown in **Figure 6**. I'll go through each of the activity types in turn and talk about what they do, but first let's see what each workflow does.

The workflow consists of a simple sequence that invokes each activity type in turn. Because the Word and Excel activities check the input document types, they won't try to process the wrong type.

The flowchart workflow in **Figure 6** uses a Switch activity to decide which Office activity should be invoked. It uses a simple expression to make this determination:

```
Right(Fields("InputDocument"), 4)
```

The cases docm and docx are directed to Word, while xlsx and xlsm are directed to Excel.

I'll use **Figure 5** to describe the actions of each activity, but the action in **Figure 6** is similar.

Data Entry

At the top of **Figure 5**, you can see an instance of the DataEntryActivity class. The DataEntryActivity presents a WPF window with a DataGrid control, which is populated by extracting the named fields from the InputDocument, which in this case is selected by the user. The control allows the user to select a document, regardless of whether one was initially provided. The user can then enter or modify the values in the fields. A separate ObservableCollection class is provided to enable the required TwoWay binding to the DataGrid, as shown in **Figure 7**.

The InputDocument can be either of the supported Office document types, Word (.docx or .docm) or Excel (.xlsx or .xlsm). To extract

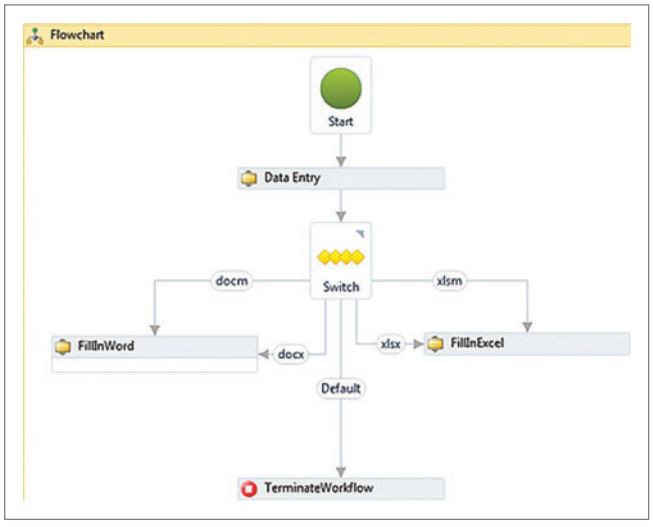
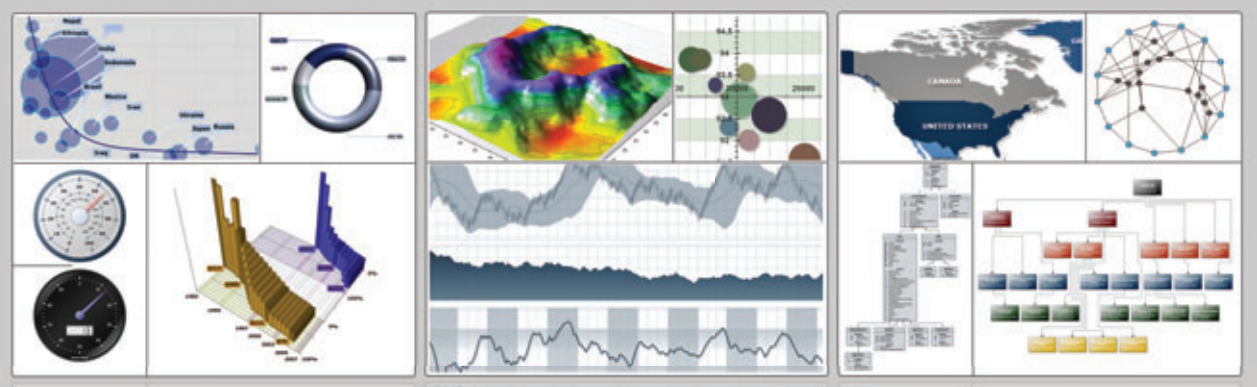


Figure 6 **Flowchart Workflow with Office Integration**

NEVRON
visualize your success

Let us help you visualize your success

Nevron provides the essential components for the creation of advanced digital dashboards, scientific and financial applications, diagrams and MMI interfaces for a variety of .NET centric technologies.



Nevron components integrate seamlessly in web and desktop 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.

DEVELOPERS

Nevron .NET Vision incorporates components that help you create enterprise grade digital dashboards, scorecards, diagrams, maps, MMI interfaces and much more.

- Chart for .NET
- Diagram for .NET
- Gauge for .NET
- Map for .NET
- User Interface for .NET

IT PROFFESIONALS

Nevron Reporting Services Vision instantly enhances your SQL Server Reporting Services 2005/2008 reports with the industry leading data visualization technology.

- Chart for SQL Server Reporting Services
- Gauge for SQL Server Reporting Services

Nevron SharePoint Vision instantly converts your SharePoint pages into advanced dashboards and reports, that unite powerful data analysis with industry leading data visualization.

- Chart for SharePoint
- Gauge for SharePoint

Make sure that your data is making the visual statement it deserves by downloading your free evaluation copy from www.nevron.com today.

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

www.nevron.com | sales@nevron.com | 1888 201 6088

the fields from the document, the appropriate OfficeInterface class is called. It loads the target document as an OpenXML object and enumerates the fields (and their contents, if present). The document formats that contain macros are supported, and the macros are carried over to the output document.

One of the fields provided by the DataEntryActivity is the TargetActivity field. This is just the name of an activity whose Fields property is to be populated with the fields collected from the input document. The TargetActivity field is used by the other activities as a way to determine whether to process the data fields.

WordFormActivity

The WordFormActivity operates on a word-processing (Word) document. It matches the Fields entries to any bookmarks in the Word document with the same names. It then inserts the value of the field into the Word bookmark. This activity will accept Word documents with (.docm) or without (.docx) macros.

ExcelFormActivity

The ExcelFormActivity operates on a spreadsheet (Excel) document. It matches the Fields entries to any simple named ranges in the Excel document with the same names. It then inserts the value of the field into the Excel named range. This activity will accept Excel documents with (.xlsm) or without (.xlsx) macros.

Excel document types have some additional special features that require special handling if filled-in data is to be formatted and handled correctly. One of these features is the variety of implicit date and time formats. Fortunately, these are well-documented (see ECMA-376 Part 1, 18.8.30 at bit.ly/fUUJ). When an ECMA format is encountered, it needs to be translated to the corresponding .NET format. For example, the ECMA format mm-dd-yy becomes M/dd/yyy.

In addition, spreadsheets have a concept of shared strings, and special handling is required when inserting a value into a cell that is to contain a shared string. The InsertSharedStringItem method used here is derived from the one contained in the OpenXML SDK:

```
If TargetCell.DataType.HasValue Then
  Select Case TargetCell.DataType.Value
    Case CellValues.SharedString    ' Handle case of a shared string data type
      ' Insert the text into the SharedStringTablePart.
      Dim index As Integer = InsertSharedStringItem(NewValue, SpreadSheetDocument)
      TargetCell.CellValue.Text = index.ToString
      Status = True
    End Select
End If
```

Figure 7 **ObservableCollection for Displaying Fields**

```
Imports System.Collections.ObjectModel

' ObservableCollection of Fields for display in WPF

Public Class WorkflowFields
  Inherits ObservableCollection(Of WorkFlowField)

  'Create the ObservableCollection from an input Dictionary

  Public Sub New(ByVal data As Dictionary(Of String, String))
    For Each Item As KeyValuePair(Of String, String) In data
      Me.Add(New WorkFlowField(Item))
    Next
  End Sub

  Public Function ContainsKey(ByVal key As String) As Boolean
    For Each item As WorkFlowField In Me.Items
      If item.Key = key Then Return True
    Next
    Return False
  End Function

  Public Shadows Function Item(ByVal key As String) As WorkFlowField
    For Each Field As WorkFlowField In Me.Items
      If Field.Key = key Then Return Field
    Next
    Return Nothing
  End Function
End Class

Public Class WorkFlowField
  Public Sub New(ByVal item As KeyValuePair(Of String, String))
    Me.Key = item.Key
    Me.Value = item.Value
  End Sub
  Property Key As String
  Property Value As String
End Class
```

Finishing Touches

The example workflow simply announces its own completion. The activity that's selected, either by document type or by the TargetActivity field, creates its output document in the specified location. From here it can be picked up by other activities for additional processing. For demonstration purposes, each of the activities ends by launching the output document, relying on Windows to open it in the appropriate application:

```
System.Diagnostics.Process.Start(OutputDocumentName)
```

If you want to print instead, you can use the following:

```
Dim StartInfo As New System.Diagnostics.ProcessStartInfo( _
  OutputDocumentName) StartInfo.Verb = "print"
System.Diagnostics.Process.Start(StartInfo)
```

Because we're working with only the document format, there's still no necessity for the workflow application to be aware of the Office version installed!

In a real production environment, more work would usually follow. Database entries might be made, and documents might end up being routed via e-mail or printed and sent to a customer. Production workflow applications would also be likely to take advantage of other services, such as persistence and tracking.

Wrapping Up

I've outlined a basic design approach to interfacing Window Workflow Foundation 4 with Office documents using the OpenXML SDK. The sample workflow illustrates this approach, and shows how it can be implemented in a way that customizes Office documents using data in the workflow. The classes from which this solution is built are readily modifiable and extensible to meet a wide variety of similar needs. Though the workflow activities are written to take advantage of WF 4 and the .NET Framework 4, the Office interface libraries are also compatible with the .NET Framework 3.5. ■

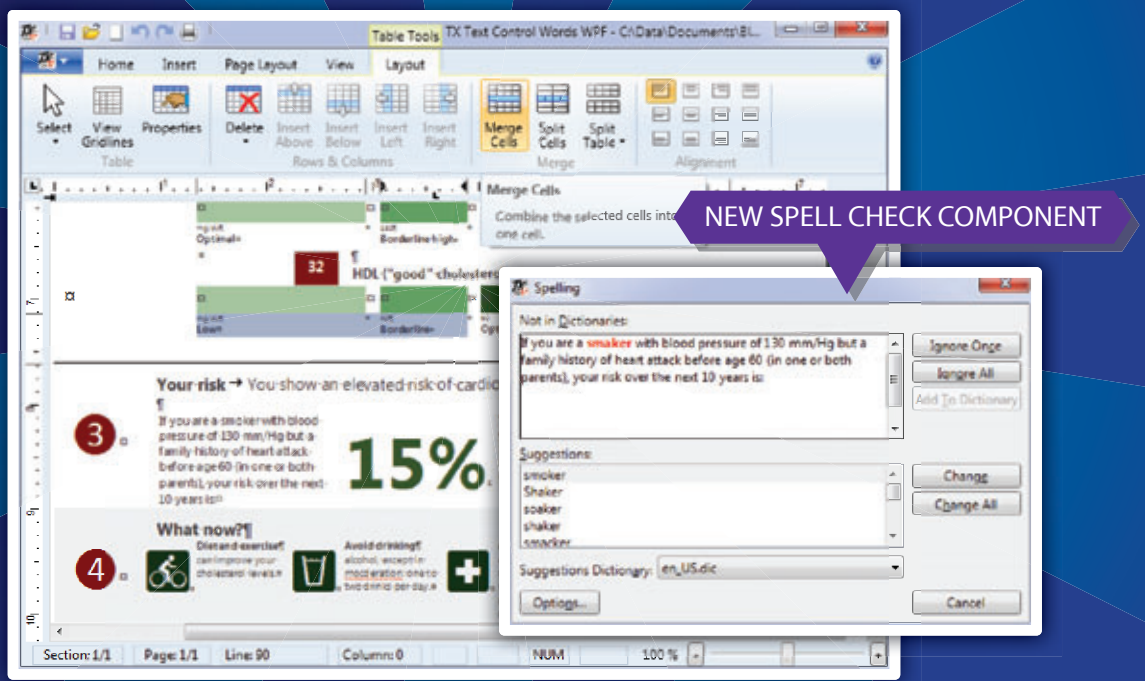
RICK SPIEWAK *is a lead software systems engineer with The MITRE Corp. He works with the U.S. Air Force Electronic Systems Center on Mission Planning. Spiewak has worked with Visual Basic since 1993 and the Microsoft .NET Framework since 2002, when he was a beta tester for Visual Studio .NET 2003. He has long experience integrating Office applications with a variety of workflow tools.*

THANKS to the following technical expert for reviewing this article:
Andrew Coates

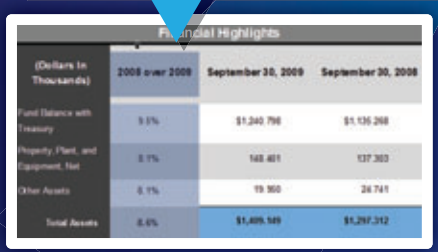
WINDOWS FORMS | WPF | ASP.NET | ACTIVEX

WORD PROCESSING COMPONENTS

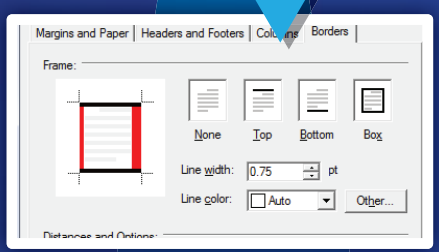
VERSION 17.0 RELEASED



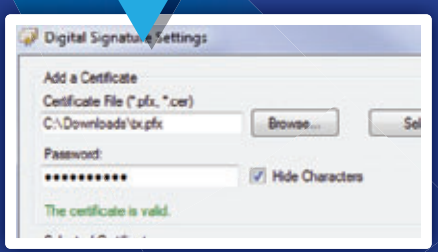
CELL MERGING, COL SELECTION



PAGE BORDERS



DIGITAL SIGNATURES IN PDF



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

Context-Aware Dialogue with Kinect

Leland Holmquest

Meet Lily, my office assistant. We converse often, and at my direction Lily performs common business tasks such as looking up information and working with Microsoft Office documents. But more important, Lily is a *virtual* office assistant, a Microsoft Kinect-enabled Windows Presentation Foundation (WPF) application that's part of a project to advance the means of context-aware dialogue and multimodal communication.

Before I get into the nuts-and-bolts code of my app—which I developed as part of my graduate work at George Mason University—I'll explain what I mean by context-aware dialogue and multimodal communication.

This article discusses:

- The meaning of context-aware dialogue and multimodal communication
- Kinect architecture
- The SpeechTracker class
- Using a dictionary to choose the right user intention
- Running the program
- The Confidence property
- Creating the illusion of a human assistant

Technologies discussed:

Microsoft Kinect

Code download available at:

code.msdn.microsoft.com/mag201204Kinect

Context-Aware Dialogue and Multimodal Communication

As human beings, we have rich and complex means of communicating. Consider the following scenario: A baby begins crying. When the infant notices his mother is looking, he points at a cookie lying on the floor. The mother smiles in that sympathetic way mothers have, bends over, picks up the cookie and returns it to the baby. Delighted at the return of the treasure, the baby squeals and gives a quick clap of its hands before greedily grabbing the cookie.

This scene describes a simple sequence of events. But take a closer look. Examine the modes of communication that took place. Consider implementing a software system where either the baby or the mother is removed and the communication is facilitated by the system. You can quickly realize just how complex and complicated the communication methods employed by the two actors really are. There's audio processing in understanding the baby's cry, squeal of joy and the sound of the clap of hands. There's the visual analysis required to comprehend the gestures represented by the baby pointing at the cookie, as well as inferring the mild reproach of the mother by giving the sympathetic smile. As often is the case with actions as ubiquitous as these, we take for granted the level of sophistication employed until we have to implement that same level of experience through a machine.

Let's add a little complexity to the methods of communication. Consider the following scenario. You walk into a room where several people are in the middle of a conversation. You hear a single word: "cool." The others in the room look to you to contribute. What could you offer? Cool can mean a great many things. For example, the person might have been discussing the temperature of the room. The speaker might have been exhibiting approval of something ("that car is cool"). The person could have been discussing the relations between countries ("negotiations are beginning to cool"). Without the benefit of the context surrounding that single word, one stands little chance of understanding the meaning of the word at the point that it's uttered. There has to be some level of semantic understanding in order to comprehend the intended meaning. This concept is at the core of this article.

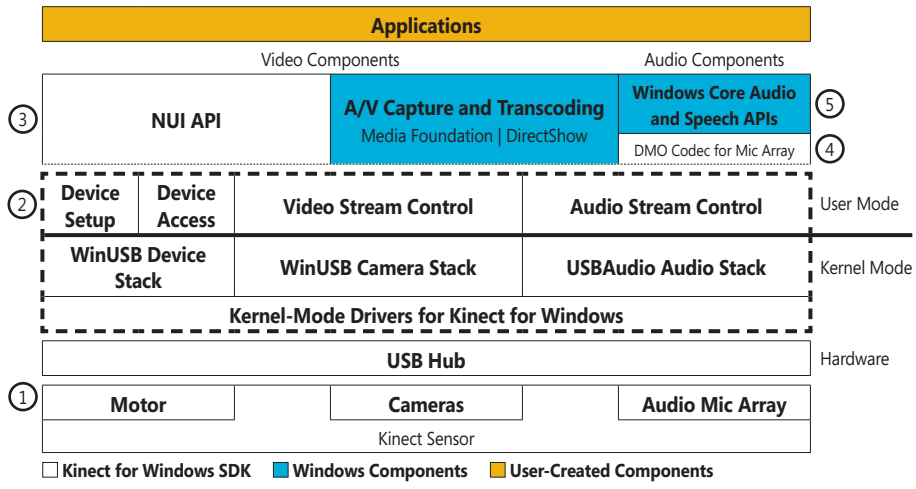


Figure 1 **Kinect for Windows Architecture**

> Lily is a virtual assistant placed in a typical business office setting.

Project Lily

I created Project Lily as the final project for CS895: Software for Context-Aware Multiuser Systems at George Mason University, taught by Dr. João Pedro Sousa. As mentioned, Lily is a virtual assistant placed in a typical business office setting. I used the Kinect device and the Kinect for Windows SDK beta 2. Kinect provides a color camera, a depth-sensing camera, an array of four microphones and a convenient API that can be used to create natural UIs. Also, the Microsoft Kinect for Windows site (microsoft.com/en-us/kinectforwindows) and Channel 9 (bit.ly/zD15UR) provide a plethora of useful, related examples. Kinect has brought incredible capabilities to developers in a (relatively) inexpensive package. This is demonstrated by Kinect breaking the Guinness World Records "fastest selling consumer device" record (on.mash.to/hVbZOA). The Kinect technical specifications (documented at bit.ly/zZ1PN7) include:

- Color VGA motion camera: 640x480 pixel resolution at 30 frames per second (fps)
- Depth camera: 640x480 pixel resolution at 30 fps
- Array of four microphones
- Field of view
  - Horizontal field of view: 57 degrees
  - Vertical field of view: 43 degrees
  - Physical tilt range: ± 27 degrees
  - Depth sensor range: 1.2m - 3.5m
- Skeletal tracking system
  - Ability to track up to six people, including two active players
  - Ability to track 20 joints per active player
- An echo-cancellation system that enhances voice input
- Speech recognition

Kinect Architecture

Microsoft also provides an understanding of the architecture that Kinect is built upon, shown in **Figure 1**.

The circled numbers in **Figure 1** correspond to the following:

1. **Kinect hardware:** The hardware components, including the Kinect sensor and the USB hub, through which the sensor is connected to the computer.
2. **Microsoft Kinect drivers:** The Windows 7 drivers for the Kinect sensor are installed as part of the beta SDK setup process as described in this document. The Microsoft Kinect drivers support:
   a. The Kinect sensor's microphone array as a kernel-mode audio device that you can access through the standard audio APIs in Windows.
   b. Streaming image and depth data.
   c. Device enumeration functions that enable an application to use more than one Kinect sensor that's connected to the computer.

```
public enum General
{
    None = 0,
    Time,
    Log,
    Schedule,
    Pulse,
    Shutdown,
    ColorViewer,
    DepthViewer,
    SkeletonViewer,
    Buttons,
    Affirmation,
    Thanks,
    Negative,
    TrackOn,
    TrackOff,
    TranslationView,
    NewsView
}

public enum Context
{
    Context = 0,
    General,
    Research,
    Office,
    Operations,
    Meeting,
    Translate,
    Entertain
}

public enum Office
{
    None = 0,
    Word,
    OpenADocument,
    Outlook,
    Excel,
    PowerPoint,
    OpenAPresentation,
    PPTNextSlide,
    PPTPreviousSlide,
    StartPresentation,
    OpenAProjectPlan
}
```

Figure 2 **Example of Context**

```
y<string, Intention> ContextPhrases = new Dictionary<string, Intention>()

eral", new Intention()                          {context= Context.General, contextOnly= true}},
{"Research", new Intention()                    {context= Context.Research, contextOnly= true}},
{"I need some information", new Intention()     {context= Context.Research, contextOnly= true}},
{"Lets Research", new Intention()               {context= Context.Research, contextOnly= true}},
{"I need to do some Research", new Intention()  {context= Context.Research, contextOnly= true}},
{"Lily, I need you to do some Research", new Intention()  {context= Context.Research, contextOnly= true}},
{"Office", new Intention()                      {context= Context.Office, contextOnly= true}},
{"Kinect", new Intention()                      {context= Context.Operations, contextOnly= true}},
{"Meeting", new Intention()                     {context= Context.Meeting, contextOnly= true}},
{"Translate", new Intention()                   {context= Context.Translate, contextOnly= true}},
{"Lily can you translate for me", new Intention()  {context= Context.Translate, contextOnly= true}},
{"Lily please translate for me", new Intention()   {context= Context.Translate, contextOnly= true}},
{"Translation", new Intention()                 {context= Context.Translate, contextOnly= true}},
{"Entertain", new Intention()                   {context= Context.Entertain, contextOnly= true}},
};
```

Figure 3 **The ContextPhrases Dictionary**

3. **NUI API:** A set of APIs that retrieves data from the image sensors and controls the Kinect devices.
4. **KinectAudio DMO:** The Kinect DMO that extends the microphone array support in Windows 7 to expose beamforming and source localization functionality.
5. **Windows 7 standard APIs:** The audio, speech and media APIs in Windows 7, as described in the Windows 7 SDK and the Microsoft Speech SDK (Kinect for Windows SDK beta Programming Guide).

In this article, I'll demonstrate how I used the microphone array and the speech recognition engine (SRE) to create vocabularies that are context-specific. In other words, the vocabularies for which Kinect is listening will be dependent on the context the user is creating. I'll show a framework in which the application will monitor what the user is doing, swapping grammars in and out of the SRE (depending on the context), which gives the user a natural and intuitive means of interacting with Lily without having to memorize specific commands and patterns of use.

The SpeechTracker Class

For Project Lily, I used a separate class called SpeechTracker to handle all voice processing. The SpeechTracker was designed to run on a thread independent of the UI, making it much more responsive, which is a critical aspect to this application. What good is an assistant who's never listening?

Prior to getting into the heart of the SpeechTracker, a few things require some forethought. The first is determining the contexts for which the application needs to listen. For example, Lily will have a "Research" context that handles actions related to looking up data and

```
Dictionary<string, Intention> GeneralPhrases = new Dictionary<string, Intention>()
{
    {"Time", new Intention()                                {context= Context.General, contextOnly= false, general= General.Time}},
    {"What is the time", new Intention()                    {context= Context.General, contextOnly= false, general= General.Time}},
    {"What time is it", new Intention()                     {context= Context.General, contextOnly= false, general= General.Time}},
    {"Lily what time is it", new Intention()                {context= Context.General, contextOnly= false, general= General.Time}},
    {"Error log", new Intention()                           {context= Context.General, contextOnly= false, general= General.Log}},
    {"Schedule", new Intention()                            {context= Context.General, contextOnly= false, general= General.Schedule}},
    {"Lily", new Intention()                                {context= Context.General, contextOnly= false, general= General.Pulse}},
    {"Lily are you there", new Intention()                  {context= Context.General, contextOnly= false, general= General.Pulse}},
    {"Lily are you listening", new Intention()              {context= Context.General, contextOnly= false, general= General.Pulse}},
    {"Lily please shut down", new Intention()               {context= Context.General, contextOnly= false, general= General.Shutdown}},
    {"Color viewer", new Intention()                        {context= Context.General, contextOnly= false, general= General.ColorViewer}},
    {"Depth viewer", new Intention()                        {context= Context.General, contextOnly= false, general= General.DepthViewer}},
    {"Skeleton viewer", new Intention()                     {context= Context.General, contextOnly= false, general= General.SkeletonViewer}},
    {"Lily, show me the color viewer", new Intention()      {context= Context.General, contextOnly= false, general= General.ColorViewer}},
    {"Lily, show me the depth viewer", new Intention()      {context= Context.General, contextOnly= false, general= General.DepthViewer}},
    {"Lily, show me the skeleton viewer", new Intention()   {context= Context.General, contextOnly= false, general= General.SkeletonViewer}},
    {"Translation viewer", new Intention()                  {context= Context.General, contextOnly= false, general= General.TranslationView}},
    {"News viewer", new Intention()                         {context= Context.General, contextOnly= false, general= General.NewsView}},
    {"Show me the buttons", new Intention()                 {context= Context.General, contextOnly= false, general= General.Buttons}},
    {"Lily, buttons please", new Intention()                {context= Context.General, contextOnly= false, general= General.Buttons}},
    {"Buttons please", new Intention()                      {context= Context.General, contextOnly= false, general= General.Buttons}},
    {"Lily, show me the buttons", new Intention()           {context= Context.General, contextOnly= false, general= General.Buttons}},
    {"Lily, can you show me the buttons", new Intention()   {context= Context.General, contextOnly= false, general= General.Buttons}},
    {"Lily, good job", new Intention()                      {context= Context.General, contextOnly= false, general= General.Affirmation}},
    {"good job Lily", new Intention()                       {context= Context.General, contextOnly= false, general= General.Affirmation}},
    {"awesome Lily", new Intention()                        {context= Context.General, contextOnly= false, general= General.Affirmation}},
    {"awesome job Lily", new Intention()                    {context= Context.General, contextOnly= false, general= General.Affirmation}},
    {"Thank you Lily", new Intention()                      {context= Context.General, contextOnly= false, general= General.Thanks}},
    {"Thanks Lily", new Intention()                         {context= Context.General, contextOnly= false, general= General.Thanks}},
    {"No Lily", new Intention()                             {context= Context.General, contextOnly= false, general= General.Negative}},
    {"Thats not right Lily", new Intention()                {context= Context.General, contextOnly= false, general= General.Negative}},
    {"Lily thats not right", new Intention()                {context= Context.General, contextOnly= false, general= General.Negative}},
    {"Thats not what I meant Lily", new Intention()         {context= Context.General, contextOnly= false, general= General.Negative}},
    {"That sucks Lily", new Intention()                     {context= Context.General, contextOnly= false, general= General.Negative}},
    {"Lily track off", new Intention()                      {context= Context.General, contextOnly= false, general= General.TrackOff}},
    {"Lily please stop tracking", new Intention()           {context= Context.General, contextOnly= false, general= General.TrackOff}},
    {"Lily turn tracking off", new Intention()              {context= Context.General, contextOnly= false, general= General.TrackOff}},
    {"Lily please turn tracking off", new Intention()       {context= Context.General, contextOnly= false, general= General.TrackOff}},
    {"Lily track on", new Intention()                       {context= Context.General, contextOnly= false, general= General.TrackOn}},
    {"Lily please start tracking", new Intention()          {context= Context.General, contextOnly= false, general= General.TrackOn}},
    {"Lily turn tracking on", new Intention()               {context= Context.General, contextOnly= false, general= General.TrackOn}},
    {"Lily please turn tracking on", new Intention()        {context= Context.General, contextOnly= false, general= General.TrackOn}},
};
```

Figure 4 **The GeneralPhrases Dictionary**

Figure 5 **Creating Grammars and Loading the Speech Recognition Engine**

```
// And finally create our Grammars
gContext = new Grammar(gbContext);
gGeneral = new Grammar(gbGeneral);
gResearch = new Grammar(gbResearch);
gOffice = new Grammar(gbOffice);
gOperations = new Grammar(gbOperations);
gMeeting = new Grammar(gbMeeting);
gTranslation = new Grammar(gbTranslation);
gEntertain = new Grammar(gbEntertain);

// We're only going to load the Context and General grammars at this point
sre.LoadGrammar(gContext);
sre.LoadGrammar(gGeneral);

allDicts = new List<Dictionary<string, Intention>>() { ContextPhrases,
                                                       GeneralPhrases,
                                                       ResearchPhrases,
                                                       OfficePhrases,
                                                       OperationsPhrases,
                                                       MeetingPhrases,
                                                       TranslationPhrases,
                                                       EntertainPhrases };
```

information; an "Office" context that handles actions such as opening Word documents and PowerPoint presentations as well as other tasks relevant to an office setting; an "Operations" context that allows the user to make adjustments to the Kinect device; and a "General" context that handles all of the little things that can come up independent of any real context (for example, "What time is it?" "Shut down" or feedback to the system). There are a few other contexts I created in my example, but one other that's important is the "Context" context. This context is used to communicate what frame of reference Lily should be in. In other words, the Context enum is used to determine the context for which the system should be listening. More on this later.

Each context is then modeled through the use of enums, as illustrated in **Figure 2**.

With the contexts modeled, the next thing needed is a means to convey what intention a user has within a specified context. In order to achieve this, I used a struct that simply contains a holder for each context and a bool:

```
struct Intention
{
  public Context context;
  public General general;
  public Research research;
  public Office office;
  public Operations kinnyops;
  public Meeting meeting;
  public Translate translate;
  public Entertain entertain;
  public bool contextOnly;
}
```

The bool is of significance. If the Intention is simply to execute a changing of context, then contextOnly is true—otherwise it's false. The utility of this will be clearer later on; for now, know that it's a necessary flag.

What's the User's Intention?

Now that Project Lily has the ability to communicate an Intention, it needs to know when and what Intention to use at run time. In order to do this, I created a System.Collections.Generic.Dictionary<TKey, TValue> Dictionary in which the key is a spoken word or phrase and the value is the associated Intention. As of the Microsoft .NET Framework 3.0, we have a succinct way to create objects and initialize properties, as shown in **Figure 3**.

This particular Dictionary defines the Context phrases. The keys are the words and phrases the end user speaks that are then associated with an Intention. Each Intention is declared and its properties set in a single line. Notice that a single Intention (for example, Research) can be mapped to multiple single words and phrases. This provides the ability to create rich vocabularies that can model the domain-specific language and lexicon of the subject at hand. (Notice one important thing about the ContextPhrases: The property contextOnly is set to true. This tells the system that this action is used only to change out what context is active. This enables the system to short-cycle the logic behind handling spoken events.)

For a better example of this, look at the snippet from the GeneralPhrases dictionary shown in **Figure 4**.

Notice that in several cases the same Intention is modeled with different phrases, giving the system the capability to handle dialogue with the user in a rich and humanistic way. One limitation you need to be aware of is that any dictionary that's going to be used by the SRE can contain no more than 300 entries. Therefore, it's wise to model one's vocabularies and grammars prudently. Even if this were not an imposed limitation, it should be considered a best practice to keep the dictionaries as light as possible for best performance.

Now that the vocabularies are mapped to Intentions, I can get to the fun part. First, the system needs to get a handle to the SpeechRecognitionEngine:

```
ri = SpeechRecognitionEngine.InstalledRecognizers().Where(r => r.Id ==
  RecognizerId).FirstOrDefault();
if (ri == null)
{
  // No RecognizerInfo => bail
  return;
}
sre = new SpeechRecognitionEngine(ri.Id);
```

Now, I'll take the phrases I developed earlier and turn them into Choices:

```
// build our categories of Choices
var contextPhrases = new Choices();
foreach (var phrase in ContextPhrases)
  contextPhrases.Add(phrase.Key);
```

> Kinect has brought incredible capabilities to developers in a (relatively) inexpensive package.

I'll do this same operation for all of the phrases I created earlier. The Choices are then passed into the Append method on a GrammarBuilder. The last thing is to create the Grammar objects and load them into the SRE. To accomplish this, I simply create a new Grammar and pass in the GrammarBuilder representing the desired grammar, as shown in **Figure 5**.

Notice that there are only two Grammars, gContext and gGeneral, loaded into the SRE, but all of the grammars were added into a List of phrases. This is how I affect the context-aware portion of listening. General and context phrases need to be continually present in the SRE because they might be spoken at any time, without any

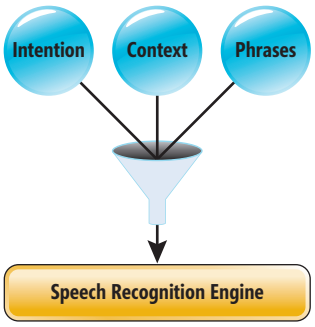


Figure 6 **Speech Recognition Engine**

predetermined pattern. However, one additional grammar will be loaded when a context phrase is identified. To complete this part of the application, I simply handle the SpeechRecognized event on the SRE. The SpeechRecognizedEventArgs argument is used to evaluate what was spoken. Referring back to **Figure 4**, if the argument has an Intention that's flagged as contextOnly, then the system needs to unload the third Grammar (if there is one) from the SRE and load the newly identified Grammar. This enables the application to listen to different vocabularies and lexicons dependent on the context that's currently in scope.

The dictionary (see **Figure 4**) has a key of type string that represents the spoken phrase and a value of type Intention that indicates what actions are intended for the system to take in order to satisfy the user's needs. Within the addition of each entry in the dictionary is the constructor for the Intention, which is typically comprised of three components: the context association; whether this is a context-changing event; and (in the case of non-context-changing) what action is intended.

With all of the dictionaries of phrases supported defined, I add this information to the SRE provided by the Kinect unit, as illustrated in **Figure 6**.

This effectively tells the SRE what to listen for and how to tag the information when it's heard. However, in an effort to make the system more intelligent and useable, I've limited the system to have only three contexts and their respective phrase dictionaries loaded in the SRE at any given time. The Context and General phrases are always loaded due to their fundamental nature. The third loaded context and phrases are determined by interaction with the end user. As Lily listens to the environment, "she" reacts to key words and phrases and removes one set of phrases to replace it with another in the SRE. An example will help clarify this.

How It Works

When Lily first starts up, ContextPhrases and GeneralPhrases are loaded into the SRE. This enables the system to hear commands that will cause either the system's context to change or will facilitate general actions. For example, after initialization is complete, should the user ask, "What time is it?" Lily "understands" this (it's part of the GeneralPhrases) and responds with the current time. Similarly, if the user says, "I need some information," Lily understands that this is a flag to load the ResearchPhrases into the SRE and begin listening for intentions mapped to the Research context. This enables Lily to achieve three important goals:

1. Lily's performance is maximized by listening only for the minimum set of phrases that could be relevant.
2. Enable use of language that would be ambiguous due to the differing meanings within different contexts (recall the "cool" example) by allowing only the lexicon of the specified context.
3. Enable the system to listen to several differing phrases but map multiple phrases to the same actions (for example, "Lily, what time is it?" "What time is it?" and "Do you know the time?" can all be mapped to the same action of telling the user the time). This potentially enables a rich lexicon for context-specific dialogue with the system. Instead of forcing a user to memorize keywords or phrases in a one-to-one fashion, mapping intended actions to a command, the designer can model several different common ways to indicate the same thing. This affords the user the flexibility to say things in normal, relaxed ways. One of the goals of ubiquitous computing is to have the devices fade into the background. Creating context-aware dialogue systems such as Lily helps remove the user's conscious awareness of the computer—it becomes an assistant, not an application.

The dictionary has a key of type string that represents the spoken phrase and a value of type Intention that indicates what actions are intended for the system to take in order to satisfy the user's needs.

Armed with all the requisite knowledge, Lily can now listen and respond with context-appropriate actions. The only thing left to do is instantiate the KinectAudioSource and specify its parameters. In the case of Project Lily, I put all of the audio processing within the SpeechTracker class. I then call the BeginListening method on a new thread, separating it from the UI thread. **Figure 7** shows that method.

Several parameters can be set, depending on the application being built. For details on these options, check out the documentation in the Kinect for Windows SDK Programming Guide (bit.ly/Avrfkd). Then

Figure 7 **KinectAudioSource**

```
Private void BeginListening()
{
  kinectSource = new KinectAudioSource());
  kinectSource.SystemMode = SystemMode.OptibeamArrayOnly;
  kinectSource.FeatureMode = true;
  kinectSource.AutomaticGainControl = true;
  kinectSource.MicArrayMode = MicArrayMode.MicArrayAdaptiveBeam;
  var kinectStream = kinectSource.Start();
  sre.SetInputToAudioStream(kinectStream,
                            new SpeechAudioFormatInfo(EncodingFormat.Pcm,
                                                      16000,
                                                      16,
                                                      1,
                                                      32000,
                                                      2,
                                                      null));
sre.RecognizeAsync(RecognizeMode.Multiple);
}
```

I simply have my WPF application register with the SpeechTracker SpeechDetected event, which is essentially a pass-through of the SRE SpeechRecognized event, but using the Intention as part of the event arguments. If the SRE finds a match of any of the context phrases it has loaded, it raises the SpeechRecognized event. The SpeechTracker handles that event and evaluates whether the Intention indicates a changing of context. If it does, the SpeechTracker handles unloading and loading the Grammars appropriately and raises the SpeechContextChanged event. Otherwise, it raises the SpeechDetected event and allows anything monitoring that event to handle it.

Figure 8 **Creating a System that Listens and Acts on Verbal Commands**

> The vocabularies for which Kinect is listening will be dependent on the context the user is creating.

The Confidence Property

One thing to note: An example I found online stated that the Confidence property in the SpeechRecognizedEventArgs is unreliable and to not use it (contradictory to the SDK documentation). I found that when I didn't use it, the SpeechRecognized event was firing fairly continually even when nothing was being spoken. Therefore, in my SpeechRecognized event handler, the very first thing I do is check the Confidence. If it isn't at least 95 percent confident, I ignore the results. (The number 95 came simply from trial and error—I can't take credit for analyzing a valid value. 95 percent just seemed to give me the level of results that I was looking for. In fact, the SDK advises to test and evaluate this value on a case-by-case basis.) When I did this, the false positives went away to 0. So I would advise testing any statements and solutions you might find online carefully. My experience was that the SDK documentation—as well as the samples that Microsoft provided at the Kinect for Windows site—was immensely valuable and accurate.

I'm frequently asked: how much speech recognition training does Kinect require? In my experience, none. Once I set the Confidence value, Kinect worked perfectly without any training, tuning and so on. I used myself as the main test subject but also included my 7- and 8-year-old daughters (thanks Kenzy and Shahrazad!), and they were thrilled to find they could tell daddy's computer what to do and it understood and acted. Very empowering to them; very rewarding for daddy!

Creating the Illusion of a Human Assistant

Being able to switch contexts based on what the system can observe of the environment creates a rich interaction between the user (human) and Lily (machine). To really enhance the illusion of having an assistant, I added a lot of little features. For example, when we human beings talk to one another, we don't necessarily repeat the same phrases over and over. Therefore, when the Intention of "pulse" is handled, the system picks out a random phrase to speak back to the user. In other words, when the user queries, "Lily?" Lily will randomly respond, "Yes," "I am here," "What can I do for you?" or a few other phrases.

I took this one step further. Some of the phrases include a holder for the user's name or gender-specific pronoun (sir or ma'am). If one of these phrases is randomly selected, then it's randomly determined whether the name or pronoun is used. This creates dialogue that's never quite the same. Little details such as this seem trivial and hardly worth the effort, but when you look at the response the human user experiences while interacting with the system, you come to realize that the uniqueness of our communications comes in part from these little details. I found myself thinking of Lily differently from other programs. As I was testing and debugging, I would need to look up some specification or some piece of code. So as I began searching I would continue talking to Lily and instruct "her" to shut down and so on. After a while, I found that I missed that level of "human interaction" when Lily was down. I think this is the best testimony I can give that Kinect brings a whole new era of natural UI.

> Creating context-aware dialogue systems such as Lily helps remove the user's conscious awareness of the computer—it becomes an assistant, not an application.

To review, **Figure 8** shows the progression of the objects needed to create a system that listens and acts on a user's verbal commands.

In the next article, I'll examine leveraging the Kinect depth- and skeleton-tracking capabilities and coupling them with the speech-recognition capabilities to achieve multimodal communication. The importance of context will become even more obvious within the multimodal communication component. You'll see how to link motions of the body to audio commands and have the system evaluate the entirety of the human communication spectrum. ■

LELAND HOLMQUEST *is an employee at Microsoft. Previously he worked for the Naval Surface Warfare Center Dahlgren. He's working on his Ph.D. in Information Technology at George Mason University.*

THANKS to the following technical expert for reviewing this article: *Russ Williams*

Batching EDI Data in BizTalk Server 2010

Mark Beckner

With the expanded Electronic Data Interchange (EDI) capabilities available in Microsoft BizTalk Server 2010, more companies are looking at how to leverage it within their environments. The batching of EDI data is some of the most valuable functionality that the platform can provide, yet it can be confusing and complex to implement. Using the steps outlined in this article, you'll learn how to quickly and easily extract data from a source database and implement mapping and batching using several scenarios. Included in the discussion is information about configuring the SQL Adapter to retrieve data using FOR XML.

Solution Overview

Working with batched data in BizTalk Server 2010 can be quite complex, but much of this complexity can be removed if you think through your architecture and decide the best place to handle the various aspects of batching. In order to understand the primary components of batching, you're going to work through the creation and configuration of each. You'll start with creating a stored procedure that extracts the source data in a format readily consumable by BizTalk—and in a format that allows for various options of how you might want to batch your data. Next, you'll look at creating a schema and options for mapping the data to the target EDI format. Finally, you'll set up batching on a BizTalk Party Agreement so the data can be created and written out to a file. In order to work through this solution, you'll need BizTalk Server 2010, Visual Studio 2010 and SQL Server 2008 R2.

This article discusses:

- Extracting data with FOR XML and XMLNAMESPACES
- Configuring a SQL Adapter
- Creating a schema with query results
- Mapping and batching options
- Batch release options

Technologies discussed:

Microsoft BizTalk Server 2010

Extracting Data from SQL Server Using FOR XML and XMLNAMESPACES

There's a powerful option when querying data from SQL Server for use in BizTalk. This option is FOR XML, which allows data to be retrieved in a specific XML format. XML is the foundation upon which all data in BizTalk is based. When extracted as XML, the data from a stored procedure can be immediately ready for consumption by orchestrations and maps without the need to generate complex artifacts that are generally required when communicating through the various SQL adapters. The FOR XML approach has some great benefits:

- It allows a SQL Server developer to write critical pieces of the integration architecture without having to learn BizTalk.
- Additional fields can be added to the results and incorporated into BizTalk components with relative ease.
- It's an efficient way to process data and it reduces the number of components that must be developed.
- It can be formatted to match target schemas in order to simplify mapping.

Figure 1 **Stored Procedure Using FOR XML and XMLNAMESPACES**

```
CREATE PROCEDURE [dbo].[GetData] AS
BEGIN
 -- define the namespace and prefix
 WITH XMLNAMESPACES('http://sql.claims.extract' as "ns0")

 -- top level is set to NULL
 SELECT NULL
       ,(SELECT ClaimData.ClaimType As [ns0:ClaimType]
               ,ClaimData.ClaimNo As [ns0:ClaimNo]
               ,ClaimData.DateFrom As [ns0:DateFrom]
               ,(SELECT first As [ns0:FName]
                       ,last As [ns0:LName]
                       ,birth As [ns0:BDate]
                FROM Members
                WHERE Members.ID = ClaimData.MemberID
                FOR XML PATH('ns0:Member'), TYPE)
         FROM ClaimData
     FOR XML PATH('ns0:Claim'), TYPE)
   FOR XML PATH('ns0:ClaimExtract'), TYPE
END
```

There are a number of options related to using FOR XML in SQL Server. The most appropriate way to use it when dealing with data created for BizTalk is to declare a specific namespace using XMLNAMESPACES and to format the XML specifically as you want it using the PATH mode (as opposed to letting SQL Server automatically name it for you, using the AUTO mode). XMLNAMESPACES is a simple clause that can be added prior to the SELECT statement that creates the XML, and the PATH mode ensures that you can create any type of hierarchy and combination of attributes and elements that you need in order to render the data in the appropriate format.

Figure 1 shows an example of a stored procedure using FOR XML PATH and XMLNAMESPACES. It first declares the namespace that the XML document will have and then formats the XML. The best approach to take when building the structure of your XML is to look at what this XML will be mapped to in BizTalk. If you can match the structure of your target schema with your source schema, the mapping between the two will be far simpler. For example, if you're creating an Employee structure to map to a target Employee structure, make the source Employee hierarchy (name, birth date, addresses, phones and so on) match the target Employee hierarchy.

In addition to formatting the data, you should consider adding business rules in the stored procedure. The more logic you can add at this level, the easier the solution will be to maintain and to develop—no need to recompile code or go through complex testing procedures. Simply update the stored procedure and test that the correct business rules are being applied to the result set. In many cases business rules can be built into the stored procedure, rather than trying to deal with them through the Business Rules Engine (BRE) in BizTalk. Creating a strong BizTalk architecture begins by ensuring that the most appropriate technology is used at each stage.

Configuring a SQL Adapter to Pull Data

Now that you have a stored procedure that returns XML, the next step is to configure BizTalk Server to retrieve the data. This can be done by using the standard SQL Adapter that ships with the product. Developers might find working with the SQL Adapter to be extremely cumbersome. In my experience it usually requires the generation of a number of schemas that are used for mapping the results to various target artifacts, and is generally neither intuitive to work with nor easy to maintain and extend. However, in some cases—like the one I'm going to look at now—it's amazingly simple to use and provides a great service.

In the case of extracting XML data straight from a stored procedure, there are several excellent benefits that using the SQL Adapter provides: It's easy to configure, requires no additional BizTalk schemas and can be configured to run on a schedule. Though you'll find it useful for the extraction of FOR XML data, in general you should use a Microsoft .NET Framework class to interact with SQL Server.

To use the SQL Adapter to call the stored procedure, begin with creating a new BizTalk Receive Location. The Receive Location should be of type "SQL," and the pipeline can be set as the standard PassThruReceive pipeline. There are several fields that deserve discussion:

- **Document Root Element Name** This is a container node for the result set. You may already have a root element defined in the XML result set from the stored procedure, but the adapter will wrap it in an additional node. This is useful if you need to split the result set into individual documents using an envelope schema—but not every solution needs to do this.
- **Document Target Namespace** This namespace is used by the schema. You can make it something similar to what you have as the namespace declared in the stored procedure.
- **SQL Command** This is the EXEC command that you use to call the stored procedure. In the case of the stored procedure outlined previously, this value would be EXEC GetData. You can easily add parameters. For example, if you wanted to call GetData for a specific Trading Partner and for a specific number of records, you could enter EXEC GetData 'TradingPartnerName', '100'.

Once the Receive Location and associated Receive Port have been created, you can create a simple Send Port that subscribes to

Figure 2 **Sample Result of SQL Adapter's Call to Stored Procedure**

```
<DataResultSet xmlns="http://SQLExtract.DataResultSet">
  <ns0:ClaimExtract xmlns:ns0="http://sql.claims.extract">
    <ns0:Claim xmlns:ns0="http://sql.claims.extract">
      <ns0:ClaimType>Institutional</ns0:ClaimType>
      <ns0:ClaimNo>ABC100</ns0:ClaimNo>
      <ns0:DateFrom>2012-01-01T00:00:00</ns0:DateFrom>
      <ns0:Member xmlns:ns0="http://sql.claims.extract">
        <ns0:FName>John</ns0:FName>
        <ns0:LName>Doe</ns0:LName>
        <ns0:BDate>1975-01-28T00:00:00</ns0:BDate>
      </ns0:Member>
    </ns0:Claim>
    <ns0:Claim xmlns:ns0="http://sql.claims.extract">
      <ns0:ClaimType>Institutional</ns0:ClaimType>
      <ns0:ClaimNo>XYZ200</ns0:ClaimNo>
      <ns0:DateFrom>2012-01-05T00:00:00</ns0:DateFrom>
      <ns0:Member xmlns:ns0="http://sql.claims.extract">
        <ns0:FName>Jane</ns0:FName>
        <ns0:LName>Doe</ns0:LName>
        <ns0:BDate>1976-10-08T00:00:00</ns0:BDate>
      </ns0:Member>
    </ns0:Claim>
  </ns0:ClaimExtract>
</DataResultSet>
```

the Receive Port and writes the XML out to a file drop. To do this, just set the BTS.ReceivePortName in the Send Port's filter to the name of the Receive Port you created for the SQL Adapter. Once everything is enabled and running, you should see an output similar to that shown in **Figure 2**.

Creating a Schema from the Query Results

The creation of the actual BizTalk XSD based on the XML retrieved from the stored procedure in the SQL Adapter can be accomplished through the use of a wizard. To create the schema, take the following steps:

1. Right-click your BizTalk Visual Studio project that you want to add the schema to and select Add | Generated Items.
2. In the window that opens, click on Generate Schemas.
3. In the Generate Schemas dialogue box, set the Document Type property to "Well-Formed XML." Note that by default, this isn't initially available. An error will pop up outlining what file needs to be run in order to enable this functionality.
4. Set the Input file to an instance of the XML shown in **Figure 3** and click OK.

These steps will add two XSDs to your project. You can decide how you want to use them, and you can also combine them into a single XSD if you prefer. An example of the top-level schema is shown in **Figure 3**.

Mapping and Batching Options

You now have your source data ready to be mapped to your EDI format. Thought needs to be put into how best to format and batch the EDI documents that are going to be sent. Some trading partners might require multiple records within a single ST/SE group, while others might require single records within this grouping. There could be limitations imposed on the total number of records within a single document or a single ST/SE, and there likely will be requirements around when a batch gets created and delivered. The 837 (Health Care Claim Specification) document format provides an excellent example. Trading partners might, for example, require a maximum of 2,500 claims be present within each ST/SE group and a maximum of 20 individual ST/SEs be present within a single document. Assessing the requirements of the various parties with which you're exchanging information will lead you to determining how best to structure your data's path through BizTalk.

As for the actual BizTalk map and associated batching, two basic options are available. One is to map your data to the target EDI schema in a single ST/SE group; the second is to map multiple records in your source to a single ST/SE group. Depending on which route you need to take, you might have to set up an envelope schema to split the source data, and you'll have some differences in your mapping and in your batch configurations.

One Record per Individual ST/SE Begin by looking at the scenario where the source data will be split and batched as single records within individual ST/SE groups. The sample source data shown in **Figure 3** has two claims in it. The goal is to take the individual claims (<ns0:Claim>) and split them out so that they can be mapped individually to the target 837 schema. Once mapped, the data can be batched as it arrives in BizTalk. In order to split the data you'll need to create an envelope schema:

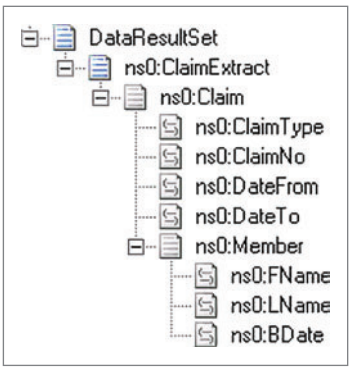


Figure 3 **The Generated XSD**

- Set the Envelope property on the schema to "Yes."
- Set the Body XPath on the root segment of your document to the node that contains the claim nodes—in this case, the ClaimExtract node.
- Deploy the envelope schema. The moment that a file is received on a Receive Location that has the default XML-Receive pipeline implemented, it will be split. There's no need to reference the envelope schema anywhere.
- If you set up a Send Port to subscribe to the Receive Port that's receiving the doc to split, the Send Port will write out individual Claim XML files. You can then use these as the source of your mapping to your target EDI schema. The Send Port should have the map of the source to the EDI transaction on it.

At this point there are many individual EDI documents stacked up in a file directory. You can now set up a second Receive Location/Send Port combination that does the batching, as follows:

- In your Receive Location, set the Receive Pipeline to EdiReceive. It will pick up the individual EDI documents that were just written out by the first Receive/Send combo.
- In your Send Port, set the filters as shown in **Figure 4**. Note that the EDI.BatchName and the EDI.DestinationPartyName are set to the information in the Party that will be configured in the next steps. This Send Port, with EDISend specified as a pipeline, can be set to write to a File Drop—the output will be the batched data.
- Create a new BizTalk Party and Agreement that represents the data exchange with the Trading Partner that will be receiving the batched documents. The main fields to configure are:
  - On the Identifiers tab, set the Sender and Receiver IDs to appropriate values. The DestinationPartyName should match what you set in the Send Port filter.
  - On the Local Host Settings and Envelopes tabs, set up the properties for the specific EDI document(s) you're handling.
  - On the Batch Configuration tab, set the following properties:
    - Set the Batch Name equal to what you set it to in the Send Port filter.
    - Set the Batch Filter with two properties:
      - BTS.ReceivePortName == [The name of your receive port where the individual EDI documents are coming in]
      - EDI.ST01 != 997
    - Set the Release option. This will most likely be based

Janus WinForms Controls Suite V4.0 | from $889.00

Janus systems

Add powerful Outlook style interfaces to your .NET applications.

- Includes Ribbon, Grid, Calendar view, Timeline and Shortcut bar
- Now features Office 2010 visual style for all controls
- Visual Studio 2010 and .NET Framework Client Profiles support
- Janus Ribbon adds Backstage Menus and Tab functionality as in Office 2010 applications
- Now features support for multiple cell selection

TX Text Control .NET for Windows Forms/WPF | from $1,045.59

TX TEXT CONTROL word processing components

Word processing components for Visual Studio .NET.

- Add professional word processing to your applications
- Royalty-free Windows Forms and WPF rich text box
- True WYSIWYG, nested tables, text frames, headers and footers, images, bullets, structured numbered lists, zoom, dialog boxes, section breaks, page columns
- Load, save and edit DOCX, DOC, PDF, PDF/A, RTF, HTML, TXT and XML

FusionCharts | from $195.02

InfoSoft Global
empowering human thoughts

Interactive Flash & JavaScript (HTML5) charts for web apps.

- Liven up your web applications using animated & data-driven charts
- Create AJAX-enabled charts with drill-down capabilities in minutes
- Export charts as images/PDF and data as CSV from charts itself
- Create gauges, dashboards, financial charts and over 550 maps
- Trusted by over 19,000 customers and 400,000 users in 110 countries

GrapeCity PowerSuite | from $1,959.02

GrapeCity PowerTools

Enterprise-ready .NET control suite for building world-class Business Applications. Includes:

- **ActiveReports** – fast, flexible and robust reporting engine
- **Spread** – full-featured Excel platform providing a complete, familiar spreadsheet experience
- **ActiveAnalysis** – interactive data analysis control providing pivoting, drill-down, drill-up, etc
- **MultiRow** – innovative Grid component allowing complete data layout customization
- **ActiveChart** – chart control providing enhanced graphical presentation of data, and more

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

VISA DISCOVER

GSA Schedule Contract GS-35F-0188R

either on schedule (deliver as many documents as have queued up over the last 24 hours) or on total number (send this batch when there are a total of 2,500 items queued). If releasing on the "Maximum number of transaction sets in" property, set the dropdown to Group.

- Once the properties have been set, click Start to activate the batching. It can take several minutes before the batch is completely started. Continue to click the Refresh button until the "Batching is activated" message appears. There should be an instance of the batch orchestration running in BizTalk (this can be seen via the Running Instances of the BizTalk Group Hub report). This orchestration is part of the BizTalk EDI Application that's installed with the EDI components in BizTalk—this isn't something you have to write.
- Click OK to save all of the settings.

Once everything is configured, enlisted, started and enabled, restart the BizTalk host instance. This will ensure everything is fresh in memory. Next, drop the individual ST/SE documents on the Receive Location and the batch will be produced once the Release mechanism triggers (for example, if you specified 2,500 as the number of transaction sets, then you must drop 2,500 individual documents).

Many Records per Individual ST/SE The next scenario to look at is mapping multiple records into individual ST/SE groups. Again, the sample source data shown in **Figure 3** has two claims in it. The new goal is to take the individual claims (the <ns0:Claim> is the root node for a single claim) and map them both to a single target 837 schema. Once mapped, the data can either be delivered as is, in a single document with a single ST/SE, or it can be batched like the previous example into a document with multiple ST/SEs, each with multiple claim records.

Start with altering the stored procedure that you wrote (see **Figure 1**). Right now, it simply brings back all of the records in the database. You need to add a parameter to limit the number of records that come back. For example, if you want to be able to put 5,000 records into a single ST/SE, then you'll need to grab only 5,000 records at a time from your source database. You can add additional parameters, such as "Trading Partner ID," to further restrict what's coming back and make the stored procedure reusable across multiple parties. Once added, you can modify the SQL Adapter settings to pass in the additional parameters. You'll eventually want to set the SQL Adapter to run on a recurring cycle (every hour, for example), extracting 5,000 records each time.

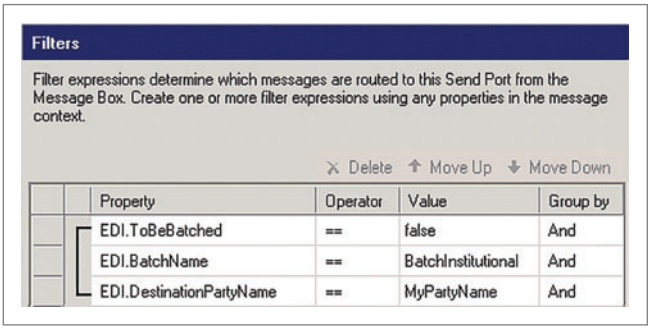


Figure 4 **Filters on the Batching Send Port**

Next, alter your mapping. Whatever mapping you put into place for the source to the target EDI schema now needs to be modified with a loop. In the case of the 837, for example, you'll want to set a loop functoid with the source being the <ns0:Claim> (from the schema in **Figure 3**) and the target being the TS837_2000A_Loop, which constitutes the top level of a target claim.

Now that your mapping is complete, you can decide whether to set up the batching within the BizTalk Party or not. If you simply want to deliver the 5,000 claims in a single ST/SE in a single document, your work is done—just set up a Send Port and ship the data out using the EDISend pipeline. If you do want to set the batching, the configuration will be the same as what you worked through earlier in this article.

Batch Release Options

You have a number of options available to release your batches. As indicated earlier, the two most common are to release batches on a specific schedule and to release when a specific number of transactions has been reached. The scheduler allows for a variety of hourly, daily and weekly configurations. For example, if you need to deliver whatever records have queued up each day at midnight, regardless of the count, you would configure the batch to release on a daily schedule at midnight. The option to release a batch based on a specific number of transactions is easily configured—just specify what the number is. Additional batching options include releasing on a specified number of characters in an interchange and on an external release trigger.

The external release trigger can be triggered using a control message dropped on the message box.

Great Simplification

As you build out your batching process in BizTalk, you should ensure you're building something that's as simple as possible. A lot of BizTalk solutions are burdened by excessive orchestrations, schemas, referenced DLLs and other artifacts. Developers often will decide that they need to create a custom batching orchestration in order to handle what they feel is a unique situation. Always focus on the long-term feasibility of your solution. Can a developer six months from now look at what you were doing and understand how to work with it? In general, you can build a process to extract data from BizTalk and deliver it to various Trading Partners with either one or no orchestration, one schema representing the source data, one schema representing the target data and a map per party. If you find yourself creating more components than this, take a step back and reanalyze your solution. You'll likely discover that you can do some great simplification. ■

MARK BECKNER *is the founder of Inotek Consulting Group LLC (inotekgroup.com). He works across the Microsoft stack, including BizTalk, SharePoint, Dynamics CRM and general .NET Framework development.*

THANKS to the following technical expert for reviewing this article: *Karthik Bharathy*

An Overview of Performance Improvements in .NET 4.5

Ashwin Kamath

On the Microsoft .NET Framework team, we've always understood that improving performance is at least as valuable to developers as adding new runtime features and library APIs. The .NET Framework 4.5 includes significant investments in performance, benefiting all application scenarios. In addition, because .NET 4.5 is an update to .NET 4, even your .NET 4 applications can enjoy many of the performance improvements to existing .NET 4 features.

When it comes to enabling developers to deliver satisfying application experiences, startup time (see msdn.microsoft.com/magazine/cc337892), memory usage (see msdn.microsoft.com/magazine/dd882521), throughput and responsiveness really matter. We set goals on improving these metrics for the different application scenarios, and then we design changes to meet or exceed them. In this article, I'll provide a high-level overview of some of the key performance improvements we made in the .NET Framework 4.5.

This article discusses the prerelease version of the Microsoft .NET Framework 4.5. All related information is subject to change.

This article discusses:

- CLR improvements
- Asynchronous programming improvements
- Parallel computing improvements
- ADO.NET improvements
- Entity Framework improvements
- Windows Communication Foundation and Windows Workflow Foundation improvements
- ASP.NET improvements

Technologies discussed:

Microsoft .NET Framework 4.5

CLR

In this release, we focused on: exploiting multiple processor cores to improve performance, reducing latency in the garbage collector and improving the code quality of native images. Following are some of the key performance improvement features.

Multicore Just-in-Time (JIT) We continually monitor low-level hardware advancements and work with chip vendors to achieve the best hardware-assisted performance. In particular, we've had multicore chips in our performance labs since they were available and have made appropriate changes to exploit that particular hardware change; however, those changes benefited very few customers at first.

At this point, nearly every PC has at least two cores, such that new features that require more than one core are immediately broadly useful. Early in the development of .NET 4.5, we set out to determine if it was reasonable to use multiple processor cores to share the task of JIT compilation—specifically as part of application startup—to speed up the overall experience. As part of that investigation, we discovered that enough managed apps have a minimum threshold number of JIT-compiled methods to make the investment worthwhile.

The feature works by JIT compiling methods likely to be executed on a background thread, which on a multicore machine will run on another core, in parallel. In the ideal case, the second core quickly gets ahead of the mainline execution of the application, so most methods are JIT compiled by the time they're needed. In order to know which methods to compile, the feature generates profile data that keeps track of the methods that are executed and then is guided by this profile data on a later run. This requirement of generating profile data is the primary way in which you interact with this feature.

With a minimal addition of code, you can use this feature of the runtime to significantly improve the startup times of both client applications and Web sites. In particular, you need to make straightforward calls to two static methods on the ProfileOptimization class, in the System.Runtime namespace. See the MSDN documentation for more information. Note that this feature is enabled by default for ASP.NET 4.5 applications and Silverlight 5 applications.

Optimized Native Images For several releases, we have enabled you to precompile code to native images via a tool called Native Image Generation (NGen). The consequent native images typically result in significantly faster application startup than is seen with JIT compilation. In this release we introduced a supplemental tool called Managed Profile Guided Optimization (MPGO), which optimizes the layout of native images for even greater performance. MPGO uses a profile-guided optimization technology, very similar in concept to multicore JIT described earlier. The profile data for the app includes a representative scenario or set of scenarios, which can be used to reorder the layout of a native image such that methods and other data structures needed at startup are colocated densely within one part of the native image, resulting in shorter startup time and less working set (an application's memory usage). In our own tests and experience, we typically see a benefit from MPGO with larger managed applications (for example, large interactive GUI applications), and we recommend its use largely along those lines.

The MPGO tool generates profile data for an intermediate language (IL) DLL and adds the profile as a resource to the IL DLL. The NGen tool is used to precompile IL DLLs after profiling, and it performs additional optimization due to the presence of the profile data. **Figure 1** visualizes the process flow.

Large Object Heap (LOH) Allocator Many .NET developers requested a solution for the LOH fragmentation issue or a way to force compaction of the LOH. You can read more about how the LOH works in the June 2008 CLR Inside Out column by Maoni Stephens at msdn.microsoft.com/magazine/cc534993. To summarize, any object of size 85,000 bytes or more gets allocated on the LOH. Currently, the LOH isn't compacted. Compacting the LOH would consume a lot of time because the garbage collector would have to move large objects and, hence, is an expensive proposition. When objects on the LOH get collected, they leave free spaces in between objects that survive the collection, which leads to the fragmentation issue.

To explain a bit more, the CLR makes a free list out of the dead objects, allowing them to be reused later to satisfy large object allocation requests; adjacent dead objects are made into one free object. Eventually a program can end up in a situation where these free memory fragments between live large objects aren't large enough to make further object allocations in the LOH, and because compaction isn't an option, we quickly run into problems. This leads to applications becoming unresponsive and eventually leads to out-of-memory exceptions.

In .NET 4.5 we've made some changes to make efficient use of memory fragments in the LOH, particularly concerning the way we manage the free list. The changes apply to both workstation and server garbage collection (GC). Please note that this doesn't change the 85,000-byte limit for LOH objects.

In .NET 4.5 we've made some changes to make efficient use of the memory fragments in the LOH, particularly concerning the way we manage the free list.

Background GC for Server In .NET 4 we enabled background GC for workstation GC. Since that time, we've seen a greater frequency of machines with upper-end heap sizes that range from a few gigabytes to tens of gigabytes. Even an optimized parallel collector such as ours can take seconds to collect such large heaps, thus blocking application threads for seconds. Background GC for server introduces support for concurrent collections to our server collector. It minimizes long blocking collections while continuing to maintain high application throughput.

If you're using server GC, you don't need to do anything to take advantage of this new feature; the server background GC will automatically happen. The high-level background GC characteristics are the same for client and server GC:

- Only full GC (generation 2) can happen in the background.
- Background GC doesn't compact.
- Foreground GC (generation 0/generation 1 GC) can happen during background GC. Server GC is done on dedicated server GC threads.
- Full-blocking GC also happens on dedicated server GC threads.

Asynchronous Programming

A new asynchronous programming model was introduced as part of the Visual Studio Async CTP and is now an important part of .NET 4.5. These new language features in .NET 4.5 enable you to productively write asynchronous code. Two new language keywords in C# and Visual Basic called "async" and "await" enable this new model. .NET 4.5 has also been updated to support asynchronous applications that use these new keywords.

Figure 1 **Process Flow with the MPGO Tool**

The Visual Studio Asynchronous Programming portal on MSDN (msdn.microsoft.com/vstudio/async) is a great resource for samples, white papers and talks on the new language features and support.

Parallel Computing Libraries

A number of improvements were made to the parallel computing libraries (PCLs) in .NET 4.5 to enhance the existing APIs.

Faster Lighter-Weight Tasks The System.Threading.Tasks.Task and Task<TResult> classes were optimized to use less memory and execute faster in key scenarios. Specifically, cases related to creating Tasks and scheduling continuations saw performance improvements of up to 60 percent.

More PLINQ Queries Execute in Parallel PLINQ falls back to sequential execution when it thinks that it would do more harm (make things slower) by parallelizing a query. These decisions are educated guesses and not always perfect, and in .NET 4.5, PLINQ will recognize more classes of queries that it can successfully parallelize.

Faster Concurrent Collections A number of tweaks were made to System.Collections.Concurrent.ConcurrentDictionary<TKey, TValue> to make it faster for certain scenarios.

For more details on these changes, check out the Parallel Computing Platform team blog at blogs.msdn.com/b/pfxteam.

ADO.NET

Null Bit Compression Row Support Null data is especially common for customers leveraging the SQL Server 2008 sparse columns feature. Customers leveraging the sparse columns feature can potentially produce result sets that contain a large number of null columns. For this scenario, row null-bit-compression (SQLNBCROW token or, simply, NBCROW) was introduced. This reduces the space used by the result set rows sent from the server with large numbers of columns by compressing multiple columns with NULL values into a bit mask. This significantly helps the tabular data stream (TDS) protocol's compression of data where there are many nulled columns in the data.

Entity Framework

Auto-Compiled LINQ Queries When you write a LINQ to Entities query today, the Entity Framework walks over the expression tree generated by the C#/Visual Basic compiler and translates (or compiles) that into SQL, as shown in **Figure 2**.

Compiling the expression tree into SQL involves some overhead, though, particularly for more complex queries. In previous versions of the Entity Framework, if you wanted to avoid having to pay this performance penalty every time a LINQ query was executed, you had to use the CompiledQuery class.

This new version of the Entity Framework supports a new feature called Auto-Compiled LINQ Queries. Now every LINQ to Entities query that you execute *automatically* gets compiled and placed in the Entity Framework query plan cache. Each additional time you run the query, the Entity Framework will find it in its query cache and won't have to go through the whole compilation process again. You can read more about it at bit.ly/iCaM2b.

```
var db = new NorthwindEntities();

var query = from c in db.Customers
            where c.Country == "USA"
            select c.CompanyName;
```

```
SELECT
[Extent1].[CompanyName] AS [CompanyName]
FROM [dbo].[Customers] AS [Extent1]
WHERE N'USA' = [Extent1].[Country]
```

Figure 2 **A LINQ to Entities Query Translated into SQL**

Windows Communication Foundation and Windows Workflow Foundation

The Windows Communication Foundation (WCF) and Windows Workflow Foundation (WF) team has also done a bunch of performance improvements in this release, such as:

- **TCP activation scalability improvements:** Customers reported an issue with TCP activation such that when many concurrent users sent requests with constant reconnections, the TCP port sharing service didn't scale well. This has been fixed in .NET 4.5.
- **Built-in GZip compression support for WCF HTTP/TCP:** With this new compression, we expect up to a 5x compression ratio.
- **Recycling host when memory usage is high for WCF:** When the memory usage is high (configurable knob), we use least recently used (LRU) logic to recycle WCF services.
- **HTTP async streaming support for WCF:** We implemented this feature in .NET 4.5 and achieved the same throughput as that of synchronous streaming but with much better scalability.
- **Generation 0 fragmentation improvements for WCF TCP.**
- **Optimized BufferManager for WCF for large objects:** For large objects, better buffer pooling has been implemented to avoid high generation 2 GC costs.
- **WF validation improvement with expression caching:** We expect up to 3x improvements for a core scenario of loading WF and executing it.
- **Implemented WCF/WF end-to-end Event Tracing for Windows (ETW):** Although this isn't a performance improvement feature, it helps customers on performance investigations.

You can find more details on the Workflow Team blog at blogs.msdn.com/b/workflowteam and in the MSDN Library article at bit.ly/n5VCtU.

ASP.NET

Improving site density (also defined as "per-site memory consumption") and cold startup time of sites in the case of shared hosting have been two key performance goals for the ASP.NET team for .NET 4.5.

In shared hosting scenarios, many sites share the same machine. In such environments traffic is usually low. Data provided by some hosting companies shows that most of the time the request per second is below 1 rps, with occasional peaks of 2 rps or more. This means that many worker processes will probably die when they're idle for a long time (20 minutes by default in IIS 7 and later). Thus startup time becomes very important. In ASP.NET, that's the time it takes a Web site to receive a request and respond to it, when the worker process was down versus when the Web site was already compiled.

Glossary of Terms

Shared Hosting: Also known as "shared Web hosting," high-density Web hosting enables hundreds—if not thousands—of Web sites to run on the same server. By sharing hardware costs, each site can be maintained for a lower cost. This technique has considerably lowered the barrier to entry for Web site owners.

Cold Startup: Cold startup is the time taken to launch when your application wasn't already present in memory. You can experience cold startup by starting an application after a system reboot. For large applications, cold startup can take several seconds because the required pages (code, static data, registry and so on) aren't present in memory and expensive disk accesses are required to bring the pages into memory.

Warm Startup: Warm startup is the time taken to start an application that's already present in memory. For example, if an application was launched a few seconds earlier, it's likely that most of the pages are already loaded in memory and the OS will reuse them, saving expensive disk-access time. This is why an application is much faster to start up the second time you run it (or why a second .NET app starts faster than the first, because parts of .NET will already be loaded in memory).

Native Image Generation, or NGen: Refers to the process of precompiling Intermediate Language (IL) executables into machine code prior to execution time. This results in two primary performance benefits. First, it reduces application startup time by avoiding the need to compile code at run time. Second, it improves memory usage by allowing for code pages to be shared across multiple processes. There's also a tool, NGen.exe, that creates native images and installs them into the Native Image Cache (NIC) on the local computer. The runtime loads native images when they're available.

Profile Guided Optimization: Profile guided optimization has been proven to enhance the startup and execution times of native and managed applications. Windows provides the toolset and infrastructure to perform profile guided optimization for native assemblies, whereas the CLR provides the toolset and infrastructure to perform profile guided optimizations for managed assemblies (called Managed Profile Guided Optimization, or MPGO). These technologies are used by many teams within Microsoft to improve the performance of their applications. For example, the CLR performs profile guided optimization of native assemblies (C++ profile guided optimization) and managed assemblies (using MPGO).

Garbage Collector: The .NET runtime supports automatic memory management. It tracks every memory allocation made by the managed program and periodically calls a garbage collector that finds memory no longer in use and reuses it for new allocations. An important optimization the garbage collector performs is that it doesn't search the whole heap every time, but rather partitions the heap into three generations (generation 0, generation 1 and generation 2). For more information on the garbage collector, please read the June 2009 CLR Inside Out column at msdn.microsoft.com/magazine/dd882521.

Compacting: In the context of garbage collection, when the heap reaches a state where it's sufficiently fragmented, the garbage collector compacts the heap by moving live objects close to each other. The primary goal of compacting the heap is to make larger blocks of memory available in which to allocate more objects.

We implemented several features in this release to improve start-up time for the shared hosting scenarios. The features used are:

- **Bin assemblies interning (sharing common assemblies):** The ASP.NET shadow copy feature enables assemblies that are used in an application domain to be updated without unloading that AppDomain (necessary because the CLR locks assemblies that are being used). This is done by copying application assemblies to a separate location (either a default CLR-determined location or a user-specified one) and loading the assemblies from that location. This allows the original assembly to be updated while the shadow copy is locked. ASP.NET turns on this feature by default for Bin folder assemblies so that DLLs can continue to be updated while a site is up and running.

  ASP.NET recognizes the Bin folder of a Web site as a special folder for compiled assemblies (DLLs) for custom ASP.NET controls, components or other code that needs to be referenced in an ASP.NET application and shared across various pages in the site. A compiled assembly in the Bin folder gets automatically referenced everywhere in the Web application. ASP.NET also detects the latest version of a specific DLL in the Bin folder for use by the Web site. Prepackaged applications intended to be used by ASP.NET sites typically install to the Bin folder rather than the Global Assembly Cache.

> GC can impact a site's memory consumption.

  The ASP.NET and CLR teams have found that when many sites reside on the same server and use the same application, many of these shadow copy DLLs tend to be exactly the same. As these files are read from disk and loaded into memory, this leads to many redundant loads that increase startup time and memory consumption. We worked on using symbolic links for the CLR to follow and then implemented identification of the common files and interned them in a special location (to which symbolic links will point). ASP.NET automatically configures shadow copying for Bin DLLs to be on. Shared hosters can now set up their machines according to the ASP.NET guidelines for maximum performance benefit.
- **Multicore JIT:** See related information in the previous "CLR" section. The ASP.NET team uses the multicore JIT feature to improve startup time by spreading the JIT compilation across processor cores. This is enabled by default in ASP.NET, so you can take advantage of this feature without any additional work. You can disable it using the following setting in your web.config file:

```
<configuration>
<!-- ... -->
<system.web>
<compilation profileGuidedOptimizations="None" />
<!-- ... -->
```

- **Prefetcher:** Prefetcher technology in Windows is very effective in reducing the disk read cost of paging during application startup. Prefetcher is now enabled (but not by default) on Windows Server as well. To enable Prefetcher for high-density Web hosting, run the following set of commands at the command line:

```
sc config sysmain start=auto

reg add "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\
Session Manager\Memory Management\PrefetchParameters" /v
EnablePrefetcher /t REG_DWORD /d 2 /f

reg add "HKEY_LOCAL_MACHINE\Software\Microsoft\
Windows NT\CurrentVersion\Prefetcher" /v
MaxPrefetchFiles /t REG_DWORD /d 8192 /f

net start sysmain
```

  You can then update the web.config file to use it in ASP.NET:

```
<configuration>
<!-- ... -->
<system.web>
<compilation enablePrefetchOptimization
  ="true" />
<!-- ... -->
```

- **Tuning GC for high-density Web hosting:** GC can impact a site's memory consumption, but it can be tuned to enable better performance. You can tune or configure GC for better CPU performance (slow down frequency of collections) or lower memory consumption (that is, more frequent collections to free up memory sooner). To enable the GC tuning, you can select the HighDensityWebHosting setting in the aspnet.config file in the Windows\Microsoft\v4.0.30319 folder in order to achieve smaller memory consumption (working set) per site:

```
<configuration>
<!-- ... -->
<runtime>
<performanceScenario
  value="HighDensityWebHosting" />
  <!-- ... -->
```

More details on ASP.NET performance improvements can be found in the "Getting Started with the Next Version of ASP.NET" white paper at bit.ly/A66I7R.

Feedback Wanted

The list presented here isn't exhaustive. There are more minor changes to improve performance that were omitted to keep the scope of this article limited to the major features. Apart from this, the .NET Framework performance teams have also been busy working on performance improvements specific to Windows 8 Managed Metro-style applications. Once you download and try the .NET Framework 4.5 and Visual Studio 11 beta for Windows 8, please let us know if you have any feedback or suggestions for the releases ahead. ■

ASHWIN KAMATH *is a program manager on the CLR team of .NET and drove the performance and reliability features for the .NET Framework 4.5. He's currently working on diagnostics features for the Windows Phone development platform.*

THANKS to the following technical experts for reviewing this article: *Surupa Biswas, Eric Dettinger, Wenlong Dong, Layla Driscoll, Dave Hiniker, Piyush Joshi, Ashok Kamath, Richard Lander, Vance Morrison, Subramanian Ramaswamy, Jose Reyes, Danny Shih and Bill Wert*

JAMES MCCAFFREY

Bacterial Foraging Optimization

Bacterial Foraging Optimization (BFO) is a fascinating artificial intelligence (AI) technique that can be used to find approximate solutions to extremely difficult or impossible numeric maximization or minimization problems. The version of BFO I describe in this article is based on the 2002 paper, "Biomimicry of Bacterial Foraging for Distributed Optimization and Control," by Dr. Kevin Passino. (This paper can be found via Internet search, but subscription is required.) BFO is a probabilistic technique that models the food-seeking and reproductive behavior of common bacteria such as E. coli in order to solve numeric optimization problems where there's no effective deterministic approach. The best way for you to get a feel for what BFO is and to see where I'm headed in this article is to examine **Figure 1** and **Figure 2**.

Figure 1 is a graph of the Rastrigin function, which is often used as a standard benchmark problem to test the effectiveness of optimization algorithms. The goal is to find the values of x and y that minimize the function. You can see that there are many valleys that are local minima solutions. However, there's only one global solution at x = 0.0 and y = 0.0 where the value of the function is 0.0.

Figure 2 is a screenshot of BFO attempting to solve the Rastrigin function. The program sets up several parameters, including the number of simulated bacteria (100 in this example). Each bacterium has a position that represents a possible solution. Initially, all bacteria are set to random positions. Each position has an associated cost that represents the value of the Rastrigin function. As the main processing loop executes, different bacteria find successively better solutions. At the end of processing, the best solution found is 0.0002 when x = -0.0010 and y = 0.0005—extremely close to but not quite the optimal solution.

In the rest of this article I'll explain in detail the BFO algorithm and walk you through the program shown running in **Figure 2**. I coded the demo program in C#, but you should be able to easily adapt the code presented here to another language such as Visual Basic .NET or Windows PowerShell. The complete source code for the program is available at code.msdn.microsoft.com/mag201203TestRun. This article assumes you have intermediate or advanced coding skills with a modern procedural language but doesn't assume you know anything about BFO or related AI techniques.

Real Bacteria Behavior

Bacteria such as E. coli are among the most successful organisms on the planet. Bacteria have semi-rigid appendages called flagella. When all flagella rotate in a counterclockwise direction, a propeller effect is created and a bacterium will swim in a more-or-less straight direction.

Code download available at code.msdn.microsoft.com/mag201204TestRun.

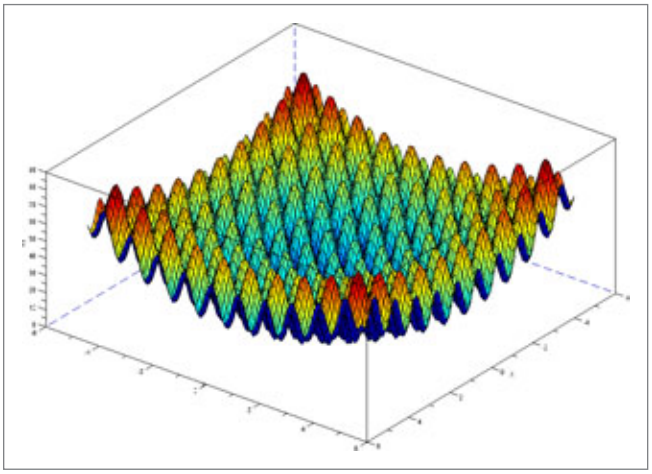


Figure 1 **The Rastrigin Function Problem**

Bacteria tend to swim when they're improving in some sense, such as when finding an increasing nutrient gradient, for example. When all flagella rotate in a clockwise direction, a bacterium will tumble quickly and point in a new direction. Bacteria tend to tumble when they encounter a noxious substance or when they're in a gradient that's not improving. Bacteria reproduce about every 20 minutes or so by asexually dividing into two identical daughters. Healthier bacteria tend to reproduce more than less-healthy bacteria.

Overall Program Structure

The overall program structure for the BFO demo is listed in **Figure 3**.

I used Visual Studio to create a C# console application named BacterialForagingOptimization. I renamed the file Program.cs to BacterialForagingOptimizationProgram.cs and deleted all template-generated using statements except for the reference to the System namespace.

I declared a class-scope Random object named random; BFO is a probabilistic algorithm, as you'll see shortly. Inside the Main method I declared several key variables. Variable dim is the number of dimensions in the problem. Because the goal in this example is to find x and y for the Rastrigin function, I set dim to 2. The minValue and maxValue variables establish arbitrary limits for both x and y. Variable S is the number of bacteria. I used slightly nondescriptive variable names such as S and Nc because these are the names used in the research article, so you can more easily use that article as a reference.

Variable Nc is the number of so-called chemotactic steps. You can think of this as a counter that represents the lifespan of each bacterium. Variable Ns is the maximum number of times a bacterium will swim in the same direction. Variable Nre is the number

of reproduction steps. You can think of this as the number of generations of bacteria. Variable Ned is the number of dispersal steps. Every now and then the BFO algorithm randomly disperses some bacteria to new positions, modeling the effects of the external environment on real bacteria. Variable Ped is the probability of a particular bacterium being dispersed. Variable Ci is the basic swim length for each bacterium. When swimming, bacteria will move no more than Ci in any single step. Variable t is a time counter to track BFO progress. Because BFO is relatively new, very little is known about the effects of using different values for BFO parameters.

The main BFO algorithm processing consists of several nested loops. Unlike most AI algorithms such as genetic algorithms that are controlled by a single time counter, BFO is controlled by multiple loop counters.

The program uses a static Cost function. This is the function that you're trying to minimize or maximize. In this example the Cost function is just the Rastrigin function. The input is an array of double that represents a bacterium's position, which in turn represents a possible solution. In this example the Cost function is defined as:

```
double result = 0.0;
for (int i = 0; i < position.Length; ++i) {
  double xi = position[i];
  result += (xi * xi) - (10 * Math.Cos(2 * Math.PI * xi)) + 10;
}
return result;
```

You can find more information about the Rastrigin function by doing an Internet search, but the point is that the Cost function accepts a bacterium's position and returns the value you're trying to minimize or maximize.

The Bacterium and Colony Classes

The BFO program defines a Colony class, which represents a collection of bacteria, with a nested Bacterium class that defines an individual bacterium. The nested Bacterium class is listed in **Figure 4**.

Class Bacterium derives from the IComparable interface so that two Bacterium objects can be sorted by their health when determining which bacteria will survive to the next generation.

The position field represents a solution. The cost field is the cost associated with the position. Field prevCost is the cost associated with a bacterium's previous position; this allows a bacterium to know if it's improving or not, and therefore whether it should swim or tumble. The health field is the sum of the accumulated costs of a bacterium during the bacterium's life span. Because the goal is to minimize cost, small values of health are better than large values.

The Bacterium constructor initializes a Bacterium object to a random position. The cost field isn't explicitly set by the constructor. The CompareTo method orders Bacterium objects from smallest health to largest health.

Figure 5 shows the simple Colony class.

The Colony class is essentially a collection of Bacterium objects. The Colony constructor creates a collection of Bacterium where each Bacterium is assigned a random position by calling the Bacterium constructor.

The Algorithm

After setting up the BFO variables such as S and Nc, the BFO algorithm initializes the bacteria colony like so:

```
Console.WriteLine("\nInitializing bacteria colony");
Colony colony = new Colony(S, dim, minValue, maxValue);
for (int i = 0; i < S; ++i) {
  double cost = Cost(colony.bacteria[i].position);
  colony.bacteria[i].cost = cost;
  colony.bacteria[i].prevCost = cost;
}
...
```

Because the Cost function is external to the colony, the cost for each Bacterium object is set outside the Colony constructor using the Cost function.

After initialization, the best bacterium in the colony is determined, keeping in mind that lower costs are better than higher costs in the case of minimizing the Rastrigin function:

```
double bestCost = colony.bacteria[0].cost;
int indexOfBest = 0;
for (int i = 0; i < S; ++i) {
  if (colony.bacteria[i].cost < bestCost) {
    bestCost = colony.bacteria[i].cost;
    indexOfBest = i;
  }
}
double[] bestPosition = new double[dim];
colony.bacteria[indexOfBest].position.CopyTo(bestPosition, 0);
Console.WriteLine("\nBest initial cost = " + bestCost.ToString("F4"));
...
```

Next, the multiple loops of the main BFO processing are set up:

```
Console.WriteLine("\nEntering main BFO algorithm loop\n");
int t = 0;
for (int l = 0; l < Ned; ++l)
{
  for (int k = 0; k < Nre; ++k)
  {
    for (int j = 0; j < Nc; ++j)
    {
      for (int i = 0; i < S; ++i) { colony.bacteria[i].health = 0.0; }

      for (int i = 0; i < S; ++i) // Each bacterium
      {
        ...
```

The outermost loop with index l handles the dispersal steps. The next loop with index k handles the reproduction steps. The third

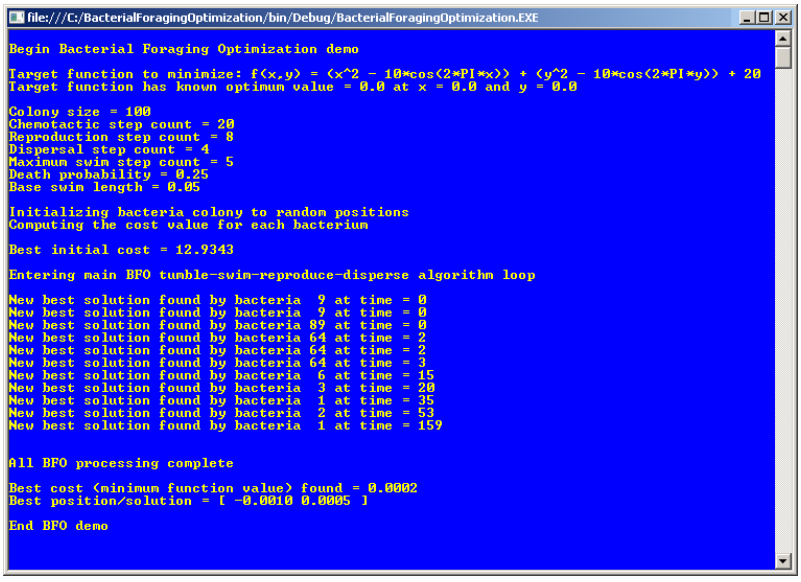


Figure 2 **Using Bacterial Foraging Optimization**

loop with index j handles the chemotactic steps that represent the lifespan of each bacterium. Inside the chemotactic loop, a new generation of bacteria has just been generated, so the health of each bacterium is reset to 0. After resetting bacteria health values inside the chemotactic loop, each bacterium tumbles to determine a new direction and then moves in the new direction, like so:

```
double[] tumble = new double[dim];
for (int p = 0; p < dim; ++p) {
  tumble[p] = 2.0 * random.NextDouble() - 1.0;
}
double rootProduct = 0.0;
for (int p = 0; p < dim; ++p) {
  rootProduct += (tumble[p] * tumble[p]);
}
for (int p = 0; p < dim; ++p) {
  colony.bacteria[i].position[p] += (Ci * tumble[p]) / rootProduct;
}
...
```

First, for each component of the current bacterium's position, a random value between -1.0 and +1.0 is generated. Then the root product of the resulting vector is computed. And then the new position of the bacterium is calculated by taking the old position and moving some fraction of the value of the Ci variable.

After tumbling, the current bacterium is updated and then the bacterium is checked to see if it found a new global best solution:

```
colony.bacteria[i].prevCost = colony.bacteria[i].cost;
colony.bacteria[i].cost = Cost(colony.bacteria[i].position);
colony.bacteria[i].health += colony.bacteria[i].cost;

if (colony.bacteria[i].cost < bestCost) {
  Console.WriteLine("New best solution found by bacteria " + i.ToString()
    + " at time = " + t);
  bestCost = colony.bacteria[i].cost;
  colony.bacteria[i].position.CopyTo(bestPosition, 0);
}
...
```

Next, the bacterium enters a swim loop where it will swim in the same direction as long as it's improving by finding a better position:

```
int m = 0;
 while (m < Ns && colony.bacteria[i].cost < colony.bacteria[i].prevCost) {
   ++m;
   for (int p = 0; p < dim; ++p) {
     colony.bacteria[i].position[p] += (Ci * tumble[p]) / rootProduct;
   }
   colony.bacteria[i].prevCost = colony.bacteria[i].cost;
   colony.bacteria[i].cost = Cost(colony.bacteria[i].position);

   if (colony.bacteria[i].cost < bestCost) {
     Console.WriteLine("New best solution found by bacteria " +
       i.ToString() + " at time = " + t);
     bestCost = colony.bacteria[i].cost;
     colony.bacteria[i].position.CopyTo(bestPosition, 0);
   }
 } // while improving
} // i, each bacterium

++t;   // increment the time counter

} // j, chemotactic loop
...
```

Variable m is a swim counter to limit the maximum number of consecutive swims in the same direction to the value in variable Ns. After swimming, the time counter is incremented and the chemotactic loop terminates.

At this point, all bacteria have lived their lifespan given by Nc and the healthiest half of the colony will live and the least-healthy half will die:

```
Array.Sort(colony.bacteria);
for (int left = 0; left < S / 2; ++left) {
  int right = left + S / 2;
  colony.bacteria[left].position.CopyTo(colony.bacteria[right].position, 0);
  colony.bacteria[right].cost = colony.bacteria[left].cost;
  colony.bacteria[right].prevCost = colony.bacteria[left].prevCost;
  colony.bacteria[right].health = colony.bacteria[left].health;
}
} // k, reproduction loop
...
```

Because a Bacterium object derives from IComparable, the Array.Sort method will automatically sort from smallest health

Figure 3 **Overall BFO Program Structure**

```
using System;
namespace BacterialForagingOptimization
{
  class BacterialForagingOptimizationProgram
  {
    static Random random = null;
    static void Main(string[] args)
    {
      try
      {
        int dim = 2;
        double minValue = -5.12;
        double maxValue = 5.12;
        int S = 100;
        int Nc = 20;
        int Ns = 5;
        int Nre = 8;
        int Ned = 4;
        double Ped = 0.25;
        double Ci = 0.05;
        random = new Random(0);
        // Initialize bacteria colony
        // Find best initial cost and position

        int t = 0;
        for (int l = 0; l < Ned; ++l) // Eliminate-disperse loop
        {
          for (int k = 0; k < Nre; ++k) // Reproduce-eliminate loop
          {
            for (int j = 0; j < Nc; ++j) // Chemotactic loop
            {
              // Reset the health of each bacterium to 0.0
              for (int i = 0; i < S; ++i)
              {
                // Process each bacterium
              }
              ++t;
            }
            // Reproduce healthiest bacteria, eliminate other half
          }

          // Eliminate-disperse
        }

        Console.WriteLine("\nBest cost found = " + bestCost.ToString("F4"));
        Console.Write("Best position/solution = ");
        ShowVector(bestPosition);
      }
      catch (Exception ex)
      {
        Console.WriteLine("Fatal: " + ex.Message);
      }
    } // Main

    static double Cost(double[] position) { ... }
  }

  public class Colony // Collection of Bacterium
  {
    // ...
    public class Bacterium : IComparable<Bacterium>
    {
      // ...
    }
  }
} // ns
```

Figure 4 **The Bacterium Class**

```
public class Bacterium : IComparable<Bacterium>
{
  public double[] position;
  public double cost;
  public double prevCost;
  public double health;
  static Random random = new Random(0);

  public Bacterium(int dim, double minValue, double maxValue)
  {
    this.position = new double[dim];
    for (int p = 0; p < dim; ++p) {
      double x = (maxValue - minValue) * random.NextDouble() + minValue;
      this.position[p] = x;
    }
    this.health = 0.0;
  }

  public override string ToString()
  {
    // See code download
  }

  public int CompareTo(Bacterium other)
  {
    if (this.health < other.health)
      return -1;
    else if (this.health > other.health)
      return 1;
    else
      return 0;
  }
}
```

(smaller is better) to largest health so the best bacteria are in the left S/2 cells of the colony array. The weaker half of bacteria in the right cells of the colony are effectively killed by copying over the data of the better half of the bacteria array into the weaker right half. Notice this implies that the total number of bacteria, S, should be a number evenly divisible by 2.

At this point the chemotactic and reproduction loops have finished, so the BFO algorithm enters the dispersal phase:

```
    for (int i = 0; i < S; ++i) {
     double prob = random.NextDouble();
     if (prob < Ped) {
       for (int p = 0; p < dim; ++p) {
         double x = (maxValue - minValue) *
           random.NextDouble() + minValue;
         colony.bacteria[i].position[p] = x;
       }
       double cost = Cost(colony.bacteria[i].position);
       colony.bacteria[i].cost = cost;
       colony.bacteria[i].prevCost = cost;
       colony.bacteria[i].health = 0.0;
       if (colony.bacteria[i].cost < bestCost) {
         Console.WriteLine("New best solution found by bacteria " +
           i.ToString() + " at time = " + t);
         bestCost = colony.bacteria[i].cost;
         colony.bacteria[i].position.CopyTo(bestPosition, 0);
       }
     } // if (prob < Ped)
   } // for

 } // l, elimination-dispersal loop
 ...
```

Each bacterium is examined. A random value is generated and compared against variable Ped to determine if the current bacterium will be moved to a random location. If a bacterium is in fact dispersed, it's checked to see if it found a new global best position by pure chance.

At this point all loops have been executed and the BFO algorithm displays the best solution found using a program-defined helper method named ShowVector:

Figure 5 **The Colony Class**

```
public class Colony
{
  public Bacterium[] bacteria;

  public Colony(int S, int dim, double minValue, double maxValue)
  {
    this.bacteria = new Bacterium[S];
    for (int i = 0; i < S; ++i)
      bacteria[i] = new Bacterium(dim, minValue, maxValue);
  }

  public override string ToString() { // See code download }

  public class Bacterium : IComparable<Bacterium>
  {
    // ...
  }
}
```

```
    Console.WriteLine("\n\nAll BFO processing complete");
    Console.WriteLine("\nBest cost (minimum function value) found = " +
      bestCost.ToString("F4"));
    Console.Write("Best position/solution = ");
    ShowVector(bestPosition);
    Console.WriteLine("\nEnd BFO demo\n");
  }
  catch (Exception ex)
  {
    Console.WriteLine("Fatal: " + ex.Message);
  }
 } // Main
 ...
```

What's the Point?

BFO is a relatively new approach to finding approximate solutions to numerical optimization problems that can't be handled using traditional mathematical techniques. In my opinion, BFO is an alternative to genetic algorithms and particle swarm optimization. There's little research evidence to answer the question of just how effective BFO is or isn't.

How might BFO be used? There are many possibilities related to software testing and also to optimization in general. For example, imagine you're trying to predict something very difficult, such as short-term changes in the price of some stock on the New York Stock Exchange. You gather some historical data and come up with some complex math equation that relates stock price to your input data, but you need to determine the parameters of your equation. You could potentially use BFO to estimate the values of your parameters where the cost function is a percentage of incorrect predictions made by your equation.

BFO is a meta-heuristic, meaning it's just a conceptual framework that can be used to design a specific algorithm. The version of BFO I've presented here is just one of many possibilities and should be considered a starting point for experimentation rather than the final word on the topic. In fact, in order to keep the size of this article manageable, I removed a bacteria swarming feature that's presented in the original BFO research article. ■

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's worked on several Microsoft products, including Internet Explorer and MSN Search. Dr. McCaffrey is the author of ".NET Test Automation Recipes" (Apress, 2006), and can be reached at jammc@microsoft.com.*

THANKS to the following technical experts for reviewing this article:
Paul Koch, Dan Liebling, Anne Loomis Thompson and Shane Williams

JOHN PAPA

Using JsRender with JavaScript and HTML

Many development platforms use templates to reduce code and simplify maintenance, and HTML5 and JavaScript are no exception. JsRender is a JavaScript library that allows you to define a boilerplate structure once and reuse it to generate HTML dynamically. JsRender brings a new templating library to HTML5 development that has a codeless tag syntax and high performance, has no dependency on jQuery nor on the Document Object Model (DOM), supports creating custom functions and uses pure string-based rendering. This column discusses scenarios for which JsRender is ideal and demonstrates how to use its various features. All code samples can be downloaded from code.msdn.microsoft.com/mag201204ClientInsight, and JsRender can be downloaded from bit.ly/ywSoNu.

According to Boris Moore and the jQuery UI team blog, in April 2011 the team decided to put jQuery templates on hold in favor of creating a logic-less and string-based rendering library. This led Moore (one of the driving forces behind jQuery templates) to create JsRender and JsViews. These two libraries are effectively the replacements for jQuery templates. You can read more on the history of templating and how JsRender has evolved on Moore's post about the jQuery Templates and JsRender roadmap (bit.ly/AdKeDk) and the jQuery team's post (bit.ly/fhnk8A). Client Insight will explore JsRender and JsViews over the next few months.

Why Templates?

Using templates with JavaScript reduces and simplifies code. Without templates, adding a series of list items and other HTML elements for a set of data might require manipulating a browser's DOM. This is where templating using a plug-in such as JsRender can be quite useful to do the heavy lifting. For example, let's assume you retrieve a set of movies and you want to display them. You could write JavaScript and manipulate the DOM, but the following code shows that even this simple task can become difficult to read, even with the help of jQuery:

```
// Without a template
var i = 1;
$(my.vm.movies).each(function () {
  var movie = this;
  $("#movieContainer1").append(
    "<div>" + i++ + ": " + movie.name + " ("
    + movie.releaseYear + ")</div>");
});
```

The code is entirely written with JavaScript and jQuery, but it can be difficult to discern the HTML from the data from the JavaScript. Using a template, we can more easily separate the structure and eliminate most of the JavaScript code. The following template (01-with-and-without-templates-with-jquery.html in the accompanying code download) demonstrates this concept:

```
<script id="myMovieTemplate" type="text/x-jsrender ">
  <div>{{:#index+1}}: {{:name}} ({{:releaseYear}})</div>
</script>
```

Notice that the template is wrapped in a script tag, its type is set accordingly and it's given an id so it can be identified later. Rendering a template requires three aspects: a template, data and a container. The template defines how the data will be rendered and the container defines where to render it. The following code shows how to render a list of movies using the template named myMovieTemplate to an element with the id of movieContainer:

```
$("#movieContainer").html($("#myMovieTemplate").render(my.vm.movies));
```

> Using templates with JavaScript reduces and simplifies code.

This example uses jQuery to simplify the syntax. It's important to note that JsRender isn't dependent on jQuery. The code to use JsRender to render the data using the template could also be written as shown here (02-jsrender-no-jquery.html in the code download):

```
my.vm = {
  movies: my.getMovies()
};

jsviews.templates({
  movieTemplate: document.getElementById("myMovieTemplate").innerHTML
});
document.getElementById("movieContainerNojQuery").innerHTML
  = jsviews.render.movieTemplate(my.vm.movies);
```

Rendering Templates

You can render templates using JavaScript in several ways. First you'll want to define your template either as a string or in a <script> tag. The <script> tag option is nice when you want to define your templates in the HTML, give them an id and reuse them. You can also create templates from strings, which gives you the ability to create them on the fly in code or even pull them from a data store.

The render method is used to render HTML content from data using a template. A set of data can be rendered with a template declared in a <script> tag using the syntax $("#myTmpl").render(data). For example, you could render a list of movies with a template using the following code:

```
// #1: Render the my.vm data using the scriptTmpl from a script tag
var htmlString = $("#scriptTmpl").render(my.vm);
// Insert the htmlString into the DOM
$("#div1").html(htmlString);
```

Code download available at code.msdn.microsoft.com/mag201204ClientInsight.

Figure 1 **Basic JsRender Syntax**

Description	Example	Output
Value of firstName property of the data item with no encoding	`{{:firstName}}`	Madelyn
Simple object path to nested property, with no encoding	`{{:movie.releaseYear}}`	1987
Simple comparison	`{{:movie.releaseYear < 2000}}`	true
Value with no encoding	`{{:movie.name}}`	Star Wars IV: Return of the Jedi
HTML-encoded value	`{{>movie.name}}`	Star Wars: Episode VI: Return of the Jedi
HTML-encoded value	`{{html:movie.name}}`	Star Wars: Episode VI: Return of the Jedi

Figure 2 **Iterating Basics**

Description	Example	Output

Loop through each item of an array, using "for"	`{{for cast}}`
	`  <div>{{:stageName}}</div>`
	`{{/for}}`
		Landon Papa
		Ella Papa

Access the data context using #data	`{{for phone}}<div>{{:#data}}</div>{{/for}}`	555-555-1212 555-555-1212

You can also compile a template from a string using the $.templates(tmplString) function and set it to a variable. You can then render the compiled template, as shown here:

```
// #2: Compile a template from a string, return a compiled template
var tmpl2 = $.templates(tmplString);
htmlString = tmpl2.render(my.vm);
$("#div2").html(htmlString);
```

You can also compile a template from a string using the $.templates(name, template) syntax, as shown here:

```
// #3: Compile a template, name and register it
$.templates("myTmpl3", tmplString);
var htmlString = $.render.myTmpl3(my.vm);
$("#div3").html(htmlString);
```

In this example, the $.templates function compiles a template using the tmplString string and registers it as a named template. The template can then be accessed by name and rendered using the $.render.name() syntax.

The $.templates function is similar to jQuery methods such as .css, or .attrib in that it provides an alternative syntax for registering and compiling multiple templates in a single call. Instead of passing two parameters (name and templateString), you can pass just one parameter consisting of a mapping object with key/value pairs for each template that's to be registered:

```
// #4: Compile multiple templates, register them and render
var tmplString2 = "<div>*** {{:movies.length}} Total Movies ***</div>";
$.templates({
  tmpl4a: tmplString,
  tmpl4b: tmplString2
});
htmlString = $.render.tmpl4a(my.vm);
htmlString += $.render.tmpl4b(my.vm);
$("#div4").html(htmlString);
```

Every property in the object becomes a named and registered template that can be rendered. The value of the property is the string that will become the template.

You have many options to create, register and render templates. Defining templates in script tags is a common approach for most scenarios. However, creating templates from strings offers a lot of flexibility. The preceding expanded syntax provides even more flexibility for associating other features with named templates (such as declaring helper functions specific to the template). All of these examples can be found in 03-rendering-templates.html in the code download.

JsRender Fundamentals

JsRender templates consist of HTML markup plus JsRender tags, such as the {{for …}} tag or the {{: …}} tag. **Figure 1** shows the syntax for the most basic of JsRender tags: the {{: }} and {{> }} tags. All JsRender template tags are wrapped with double curly braces. The tag name (in this case the ":" or ">" character) can be followed by one or more parameters or expressions. (In the case of the {{: }} tag, the result of the expression would then be rendered.) Once a template has been defined and there's data to render in that template, it can be rendered.

The following code contains an HTML element named movieContainer (this is where the template will be rendered):

```
<div id="movieContainer" class="resultsPanel movieListItemMedium"></div>
<script id="movieTemplate" type="text/x-jsrender">
  <div class="caption">{{:name}}</div>
  <div class="caption">{{>name}}</div>
  <img src="{{:boxArt.smallUrl}}"/>
  <div class="text">Year Released: {{:releaseYear}}</div>
  <div class="text">Rating: {{:rating}}</div>
</script>
```

You have many options to create, register and render templates.

The preceding code also contains the template named movieTemplate, which defines a div to display the name of the movie using no HTML encoding using the syntax {{:name}}. The title in the sample data may contain HTML elements, so to render elements containing HTML it's important not to use encoding. However, if you want to render the encoded HTML, you can use the syntax with the > character or use HTML (as shown in **Figure 1**).

The *name* property value contains HTML elements, so for demonstration purposes only, the preceding code displays the name property's value with no encoding ({{:name}}) and then shows the HTML-encoded value ({{>name}}). You can run the full example in 04-render-values.html in the code download.

The sample movie data passed to the template has a property for boxArt, which in turn has a property for the smallUrl of the image. The img tag src is set by diving into an object graph's hierarchy using the dot syntax boxArt.smallUrl. Paths can also be traversed using square brackets, so instead of writing boxArt.smallUrl, the same result code be achieved using boxArt['smallUrl'].

It's important to note that JsRender syntax can also be used to render other values such as the names of classes or ids for HTML elements.

JsRender template rendering detects whether the data parameter is an array or not. If it's an array, the return value is the concatenation of the strings that would result from passing each of the individual array items to the render method. So the template will be rendered once for each data item and the result will be the concatenated string.

Figure 3 **Rendering an Object with and Without Encoding**

The following code demonstrates how a single movie object from the array is rendered to the movieTemplate (the full source is available in sample 04-render-values.html):

```
my.vm = { movies: getMovies() };
$("#movieContainer").html($("#movieTemplate").render(my.vm.movies[1]));
```

The next section will demonstrate how the template could be written to iterate through an array, as well.

Drilling into Hierarchical Data

Templates are often used to render a series of items, which can often contain nested and hierarchical data (object graphs). **Figure 2** shows how JsRender can iterate through a data series using the {{for}} tag. The parameter for the {{for}} tag can be an array or a series of arrays, which will be iterated.

The sample movie data defines that each movie has an array of rating stars called RatingStars. The property contains the CSS class to display a rating star for the movie. The following code (from 05-for-data.html in the sample code) demonstrates how to iterate through each item in the RatingStars array and render the CSS class name using the {{:#data}} syntax:

```
<ul>
  {{for ratingStars}}
  <li class="rating {{:#data}}"/>
  {{/for}}
</ul>
```

Figure 4 **Rendering a List of Items and Using Conditionals**

```
{{for movies}}
  <li class="movieListItemMedium">
  <div class="caption">{{:name}}</div>
  {{if boxArt.smallUrl}}
    <img src="{{:boxArt.smallUrl}}"/>
  {{else}}
    <img src="../images/icon-nocover.png"/>
  {{/if}}
  <br/>
  <div class="text">Year Released: {{:releaseYear}}</div>
  <div class="text">rating: {{:rating}}</div>
  <ul>
    {{for ratingStars}}
      <li class="rating {{:#data}}"/>
      {{/for}}
  </ul>
  <br/>
  <div class="text">${{:salePrice}}</div>
  {{if fullPrice !== salePrice}}
    <div class="text highlightText">PRICED TO SELL!</div>
  {{/if}}
  </li>
{{/for}}
```

The #data token is a JsRender keyword that represents the object being iterated. In this case, the RatingStars array contains a string array, so #data will represent a string. The output for this sample is shown in **Figure 3**, where the rating stars are displayed beside the movie's image.

Arrays to Templates

When an array is passed to a template, the template is rendered once for each item in the array. A template renders against a single data item, but if it includes a {{for}} template tag with an array parameter, then the section of the template between the opening and closing tags {{for}} and {{/for}} will be further iterated. The following code will pass an object my.vm that contains an array of movies to the render function to display a list item for each movie (the complete source can be found in the sample 06-if-else.html):

```
$("#movieList").html($("#movieTemplateMedium").render(my.vm));
```

The template will be rendered as content of the <ul> element with the id movieList:

```
<ul id="movieList"></ul>
```

Figure 4 shows the template *movieTemplateMedium* begins by rendering <li>. For every movie in the array, an <li> will be rendered under the container's <ul>.

The template in **Figure 4** receives the my.vm object as its context and then loops through the movies array property because of the {{for movies}} code. If the template received the array my.vm.movies instead of my.vm, the {{for}} {{/for}} block could be removed, as the template would then execute for every item in the array.

Traversing Paths

As we've seen previously, paths can be traversed using the dot syntax or the square brackets. However, you can also go back up an object hierarchy using #parent or identify an array element using square brackets. For example, you could replace the code that displays the img tag in the template with following:

```
<img src="{{:#parent.parent.data[2].boxArt.smallUrl}}"/>
```

Figure 5 **Traversing Paths to Display the Same Movie Image for Every Movie**

Figure 6 **Conditionals, Expressions and Operators**

Description	Example	Output

Creating an if block evaluated on a comparison	`{{if fullPrice !== salePrice}}`
	`  <div class="text highlightText">`
	`  PRICED TO SELL!</div>`
	`{{/if}}`
		PRICED TO SELL!

An if/else block	`{{if qtyInStock >= 10}}`
	`  In Stock`
	`{{else qtyInStock}}`
	`  Only {{:qtyInStock}} left!`
	`{{else}}`
	`  Not available.`
	`{{/if}}`
		Only 5 left!

Figure 7 **Counters and Conditionals in Multiple Expressions**

Description	Example	Output

Using an index to count inside of a template	`{{for movies}}`
	`<div class="caption">`
	`  {{:#index}}: {{:name}}`
	`</div>`
	`{{/for}}`
		3: The Princess Bride

Using logical operators	`{{if stars.length || cast.length}}`
	`  <div class="text">Full Cast:</div>`
	`{{/if}}`
		Full Cast:

The context for the template is an item in the my.vm.movies array. So by navigating up through the parent twice, the context will become the my.vm object. Then you can grab the movie with index 2 and use its boxArt.smallUrl property for the image. The result of this would be to display the same movie (the third movie) for every movie in the array (as shown in **Figure 5**). Not exactly ideal in this situation, but it serves the point of showing how to traverse objects both up and down the hierarchy.

Conditionals

The JsRender syntax also support conditionals using the tags {{if}} and {{else}} (shown in **Figure 6**). The {{if}} tag may be followed by zero, one or more {{else}} tags, and then a closing {{/if}} tag. The content of the block between the {{if}} tag and the {{/if}} (or up to the first {{else}} tag if one exists) will be rendered in the output only if the value of the expression in the {{if}} is "truthy."

The {{else}} tag can act like an ifelse when it includes an expression of its own. Notice the second example in **Figure 6**, which demonstrates a conditional with two else tags. This code evaluates three conditions in sequence.

The block following an {{else someCondition}} tag will be output if the condition evaluates to true (or, more precisely, truthy). This syntax can be used effectively like a switch/case statement to evaluate *n* number of conditions.

The expressions used in JsRender tags can be complex expressions using evaluation operators including paths, strings, numbers, function calls and comparison operators such as ===, !== and <=. This is the case both for the expressions evaluated and rendered by the {{: }} tag, and, here, for the conditional expressions used in the {{if}} and {{else}} tags.

Using Counters and Complex Conditional Expressions

Sometimes it's necessary to display or use a counter variable in a template. One such scenario is when you want to display a row number in the template. Another such scenario might be that you want to assign a unique id to each element in a template so you can reference it later. For example, you iterate through a list of movies and you wrap them all inside a tag. You could assign the id of the div tag to be movieDiv1, movieDiv2 and so on. Then you could find the element by its id when needed, perhaps to bind it to an event handler using the jQuery delegate function. For both of these scenarios, you can use the JsRender keyword #index (which starts at 0). As shown in **Figure 7**, #index can be displayed in a template quite easily.

Oftentimes a single expression may not be adequate. JsRender supports multiple expressions using logical operators such as || and && ("or" and "and," respectively). This makes it easy to combine multiple expressions when you want to evaluate either one of them to be true.

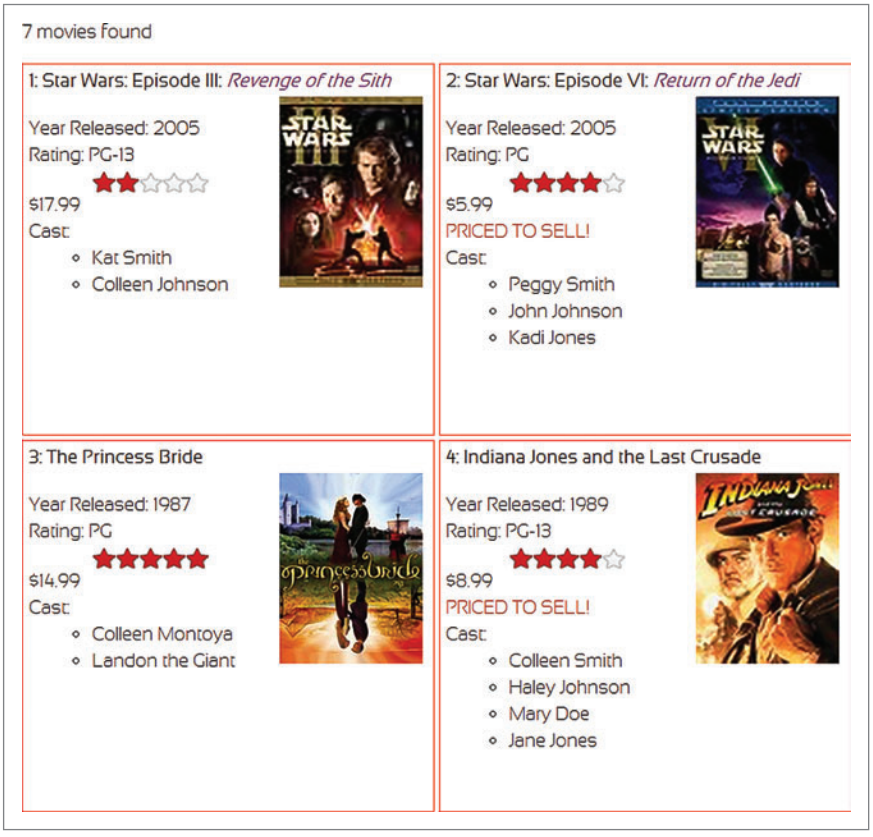


Figure 8 **Putting It All Together**

Iterate Multiple Arrays Together

The {{for}} tag can also iterate through multiple arrays at once. This is helpful when there are two arrays that need to be traversed together. For example, the following code demonstrates how the for tag will iterate through both the Stars and the Cast arrays together:

```
{{for stars cast}}
  <div class="text">{{:name}}</div>
{{/for}}
```

If there are two items in the Stars array and three in the Cast array, all five items' content will be rendered.

This is a good situation to use the #data keyword, too, if the items in the arrays aren't objects (with properties) but simple string values. For example, the arrays could be MobilePhones and HomePhones string arrays, and you might want to list them in a single list.

Nested Templates

Applications often retrieve object graphs where objects have properties whose values are arrays of objects that may also contain other arrays of objects. These scenarios can quickly become difficult to manage when nesting multiple {{for}} statements. Consider a customer object may have orders, which have order items, which have a product, which has warehouses. This hierarchical data could be rendered by using multiple {{for}} tags. This is a great situation where the content of a {{for}} block can be packaged as a separate template and rendered as a nested template. This can reduce code and make it more easily readable and maintainable, and provide for reuse and modularity of the templates.

You can use a nested template with a {{for}} tag by removing the block content (making it self-closing) and specifying an external template using the tmpl parameter. For example, you can set the tmpl parameter to be the name of a named template (registered using $.templates) or you can use a jQuery selector for a template declared in a script block. The following syntax uses the named template myTmpl for each item in the array named myArray:

```
{{for myArray tmpl="myTmpl"/}}
```

The following code snippet will use the template with the id of movieTemplateMedium for each item in the movies array (from the sample 04-tmpl-combo-iterators.html):

```
<ul>
  {{for movies
    tmpl="#movieTemplateMedium"/}}
</ul>
```

You can iterate through multiple arrays and apply a template to them, as well. For example, the following code snippet will use the castTemplate for each item in both the Stars and Cast properties (which are both arrays from sample 07-tmpl-combo-iterators.html):

```
<ul>
  {{for stars cast tmpl="#castTemplate"/}}
</ul>
```

Notice that in each of the previous code snippets the {{for}} tag is a self-closing tag rather than a block tag containing content. Instead of iterating over its block content, it's deferring the iteration to the nested template indicated by the tmpl property and is rendering that template once for each item in the array.

The {{if}} template tag can also be used as a self-closing tag referencing another template for its content, just like the {{for}} tag. For example, the following code would render the named template myAddressTmpl if *address* is truthy:

```
{{if address tmpl="myAddressTmpl"/}}
```

The samples available for download demonstrate all of the features discussed in this article. **Figure 8** shows the list of movies rendered using all of these features from the sample 07-tmpl-combo-iterators.html file.

More Under the Covers

This column demonstrates the basic features of JsRender, but there's much more under the covers. For example, although the conditional tags can contain multiple {{for}} tags with conditions (such as a switch/case statement), there might be cleaner ways to handle this situation, such as using custom tags. JsRender supports custom tags for complex logic, helper functions, navigating up the object graph using paths, customer converter functions, allowing JavaScript code within templates, encapsulation of templates and more. I'll explore some of these techniques in the next issue. ■

JOHN PAPA *is a former evangelist for Microsoft on the Silverlight and Windows 8 teams, where he hosted the popular Silverlight TV show. He has presented globally at keynotes and sessions for the BUILD, MIX, PDC, TechEd, Visual Studio Live! and DevConnections events. Papa is also a Microsoft Regional Director, a columnist for Visual Studio Magazine (Papa's Perspective) and an author of training videos with Pluralsight. Follow him on Twitter at twitter.com/john_papa.*

THANKS to the following technical expert for reviewing this article: *Boris Moore*

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

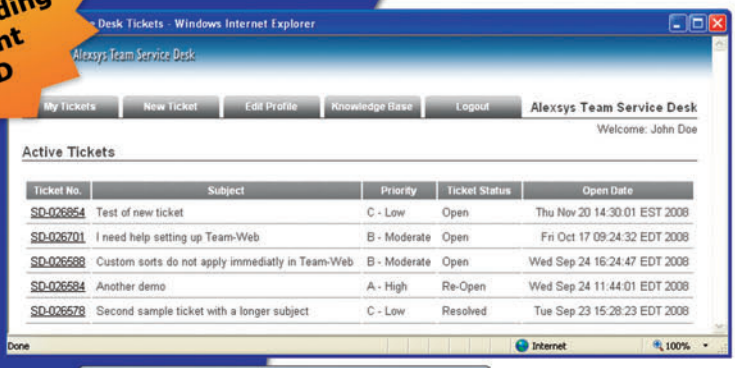
Service Desk Features

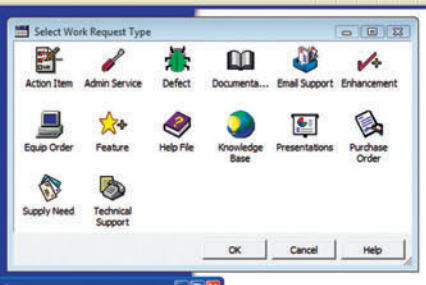
- Fully Secure
- Unlimited Users Self Registered or Active Directory
- Integrated into Your Web Site
- Fast/AJAX Dynamic Content
- Unlimited Service Desks
- Visual Service Desk Builder

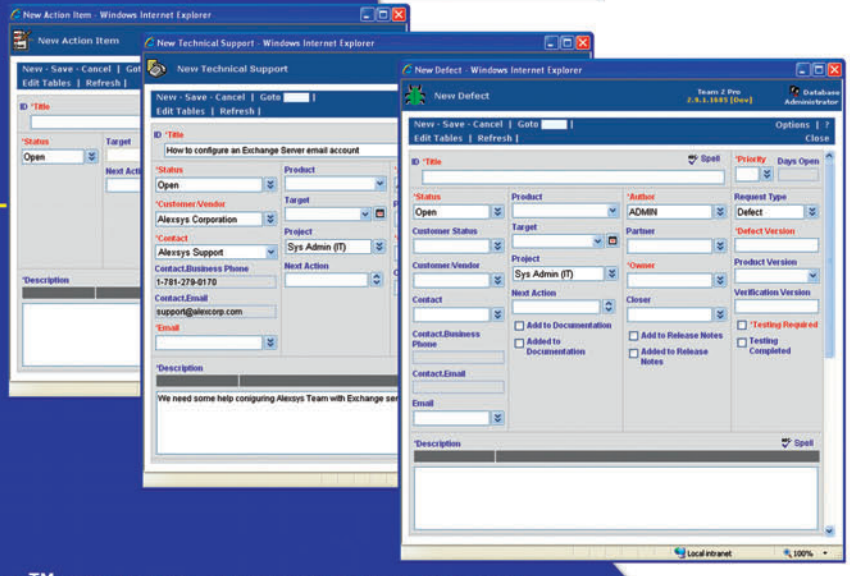
Team 2 Features

- 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.

CHARLES PETZOLD

Musical Instruments for Windows Phone

Every Windows Phone has a built-in speaker and a headphone jack, and it would surely be a shame if their use was limited to making phone calls. Fortunately, Windows Phone applications can also use the phone's audio facilities for playing music or other sounds. As I've demonstrated in recent installments of this column, a Windows Phone application can play MP3 or WMA files stored in the user's music library, or files downloaded over the Internet.

A Windows Phone app can also dynamically generate audio waveforms, a technique called "audio streaming." This is an extremely data-intensive activity: For CD-quality sound you need to generate 16-bit samples at a rate of 44,100 samples per second for both left and right channels, or a whopping 176,400 bytes per second!

But audio streaming is a powerful technique. If you combine it with multi-touch, you can turn your phone into an electronic music instrument, and what could be more fun than that?

Conceiving a Theremin

One of the very earliest electronic music instruments was created by Russian inventor Léon Theremin in the 1920s. The player of a theremin doesn't actually touch the instrument. Instead, the player's hands move relative to two antennas, which separately control the volume and pitch of the sound. The result is a spooky quivering wail that glides from note to note—familiar from movies such as "Spellbound" and "The Day the Earth Stood Still," the occasional rock band, and season 4, episode 12 of "The Big Bang Theory." (Contrary to popular belief, a theremin was not used for the "Star Trek" theme.)

Can a Windows Phone be turned into a hand-held theremin? That was my goal.

The classical theremin generates sound through a heterodyning technique in which two high-frequency waveforms are combined to produce a difference tone in the audio range. But this technique is impractical when waveforms are generated in computer software. It makes much more sense to generate the audio waveform directly.

After toying briefly with the idea of using the phone's orientation to control the sound, or for the program to view and interpret hand movements through the phone's camera à la Kinect, I settled on a much more prosaic approach: A finger on the phone's screen is a two-dimensional coordinate point, which allows a program to use one axis for frequency and the other for amplitude.

Code download available at code.msdn.microsoft.com/mag201204TouchAndGo.

Figure 1 **Amplitude and Frequency Calculation for Theremin**

```
public partial class MainPage : PhoneApplicationPage
{
  static readonly Pitch MIN_PITCH = new Pitch(Note.C, 3);
  static readonly Pitch MAX_PITCH = new Pitch(Note.C, 7);
  static readonly double MIN_FREQ = MIN_PITCH.Frequency;
  static readonly double MAX_FREQ = MAX_PITCH.Frequency;
  static readonly double MIN_FREQ_LOG2 = Math.Log(MIN_FREQ) / Math.Log(2);
  static readonly double MAX_FREQ_LOG2 = Math.Log(MAX_FREQ) / Math.Log(2);

  ...

  double xStart;      // The X coordinate corresponding to MIN_PITCH
  int xDelta;         // The number of pixels per semitone

  void OnLoaded(object sender, EventArgs args)
  {
    int count = MAX_PITCH.MidiNumber - MIN_PITCH.MidiNumber;
    xDelta = (int)((ContentPanel.ActualWidth - 4) / count);
    xStart = (int)((ContentPanel.ActualWidth - count * xDelta) / 2);

    ...

  }

  ...

  double CalculateAmplitude(double y)
  {
    return Math.Min(1, Math.Pow(10, -4 * (1 - y / ContentPanel.ActualHeight)));
  }

  double CalculateFrequency(double x)
  {
    return Math.Pow(2, MIN_FREQ_LOG2 + (x - xStart) / xDelta / 12);
  }

  ...
}
```

Doing this intelligently requires a little knowledge about how we perceive musical sounds.

Pixels, Pitches and Amplitudes

Thanks to pioneering work by Ernst Weber and Gustav Fechner in the 19th century, we know that human perception is logarithmic rather than linear. Incremental linear changes in the magnitude of stimuli are not perceived as equal. What we instead perceive as equal are changes that are *proportional* to the magnitude, often conveniently expressed as fractional increases or decreases. (This phenomenon extends beyond our sensory organs. For example, we feel that the difference between $1 and $2 is much greater than the difference between $100 and $101.)

Human beings are sensitive to audio frequencies roughly between 20Hz and 20,000Hz, but our perception of frequency is

Figure 2 **The ThereminOscillator Class**

```
public class ThereminOscillator : Oscillator
{
  readonly double ampStep;
  readonly double freqStep;

  public const double MIN_AMPLITUDE = 0.0001;

  public ThereminOscillator(int sampleRate)
    : base(sampleRate)
  {
    ampStep = 1 + 0.12 * 1000 / sampleRate;     // ~1 db per msec
    freqStep = 1 + 0.005 * 1000 / sampleRate;   // ~10 cents per msec
  }

  public double Amplitude { set; get; }
  public double DestinationAmplitude { get; set; }
  public double DestinationFrequency { set; get; }

  public override short GetNextSample(double angle)
  {
    this.Frequency *= this.Frequency < this.DestinationFrequency ?
                                  freqStep : 1 / freqStep;
    this.Amplitude *= this.Amplitude < this.DestinationAmplitude ?
                                  ampStep : 1 / ampStep;
    this.Amplitude = Math.Max(MIN_AMPLITUDE, Math.Min(1, this.Amplitude));
    return (short)(short.MaxValue * this.Amplitude * Math.Sin(angle));
  }
}
```

not linear. In many cultures, musical pitch is structured around the octave, which is a doubling of frequency. When you sing "Somewhere Over the Rainbow," the two syllables of the first word are an octave apart regardless of whether the leap is from 100Hz to 200Hz, or from 1,000Hz to 2,000Hz. The range of human hearing is therefore about 10 octaves.

The octave is called an octave because in Western music it encompasses eight lettered notes of a scale where the last note is an octave higher than the first: A, B, C, D, E, F, G, A (which is called a minor scale) or C, D, E, F, G, A, B, C (the major scale).

Due to the way these notes are derived, they are not perceptually equally distant from one another. A scale in which all the notes are equally distant requires five more notes for a total of 12 (not counting the first note twice): C, C#, D, D#, E, F, F#, G, G#, A, A# and B. Each of these steps is known as a semitone, and if they're equally spaced (as they are in common equal-temperament tuning), each note has a frequency that is the 12th root of two (or about 1.059) times the frequency of the note below it.

The semitone can be further divided into 100 cents. There are 1,200 cents to the octave. The multiplicative step between cents is the 1,200th root of two, or 1.000578. The sensitivity of human beings to changes in frequency varies widely, of course, but is generally cited to be about five cents.

This background into the physics and mathematics of music is necessary because the theremin program needs to convert a pixel location of a finger to a frequency. This conversion should be done so that each octave corresponds to an equal number of pixels. If we decide that the theremin is to have a four-octave range corresponding to the 800-pixel length of the Windows Phone screen in landscape mode, that's 200 pixels per octave, or six cents per pixel, which corresponds nicely with the limits of human perception.

The amplitude of a waveform determines how we perceive the volume, and this, too, is logarithmic. A decibel is defined as 10 times the base 10 logarithm of the ratio of two power levels. Because the power of a waveform is the square of the amplitude, the decibel difference between two amplitudes is:

$$20\log_{10}\frac{A_1}{A_0}$$

CD audio uses 16-bit samples, which allows that ratio between maximum and minimum amplitudes to be 65,536. Take the base 10 logarithm of 65,536 and multiply by 20 and you get a 96-decibel range.

One decibel is about a 12 percent increase in amplitude. Human perception to changes in amplitude is much less sensitive than to frequency. A few decibels are required before people notice a change in volume, so this can be easily accommodated on the 480-pixel dimension of the Windows Phone screen.

Making It Real

The downloadable code for this article is a single Visual Studio solution named MusicalInstruments. The Petzold.MusicSynthesis project is a DLL that mostly includes files I discussed in last month's installment of this column (msdn.microsoft.commagazine/tktktktk). The Theremin application project consists of a single landscape page.

What type of waveform should a theremin generate? In theory, it's a sine wave, but in reality it's a somewhat distorted sine wave, and if you try to research this question on the Internet, you won't

Figure 3 **The Touch.FrameReported Handler in Theremin**

```
void OnTouchFrameReported(object sender, TouchFrameEventArgs args)
{
  TouchPointCollection touchPoints = args.GetTouchPoints(ContentPanel);

  foreach (TouchPoint touchPoint in touchPoints)
  {
    Point pt = touchPoint.Position;
    int id = touchPoint.TouchDevice.Id;

    switch (touchPoint.Action)
    {
      case TouchAction.Down:
        oscillator.Amplitude = ThereminOscillator.MIN_AMPLITUDE;
        oscillator.DestinationAmplitude = CalculateAmplitude(pt.Y);
        oscillator.Frequency = CalculateFrequency(pt.X);
        oscillator.DestinationFrequency = oscillator.Frequency;
        HighlightLines(pt.X, true);
        touchID = id;
        break;

      case TouchAction.Move:
        if (id == touchID)
        {
           oscillator.DestinationFrequency = CalculateFrequency(pt.X);
           oscillator.DestinationAmplitude = CalculateAmplitude(pt.Y);
           HighlightLines(pt.X, true);
        }
        break;

      case TouchAction.Up:
        if (id == touchID)
        {
          oscillator.DestinationAmplitude = 0;
          touchID = Int32.MinValue;

          // Remove highlighting
          HighlightLines(0, false);
        }
        break;
    }
  }
}
```

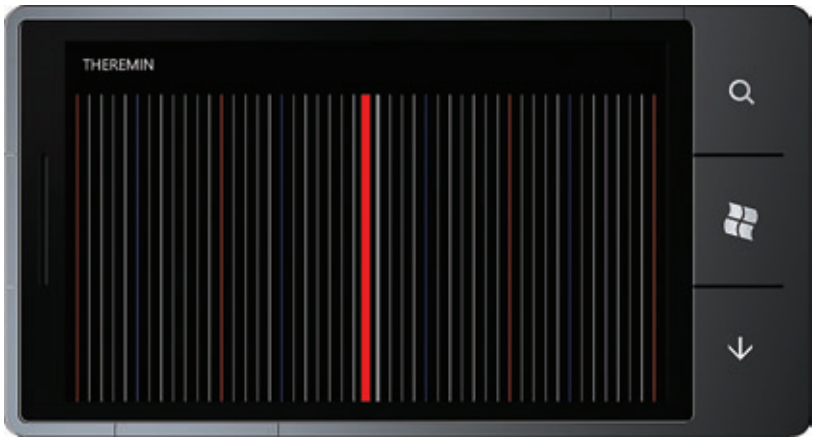


Figure 4 **The Theremin Display**

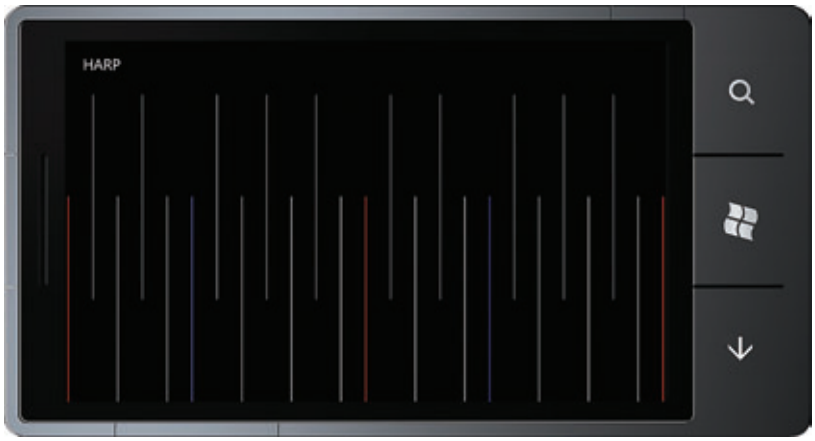


Figure 5 **The Harp Program**

find a lot of consensus. For my version, I stuck with a straight sine wave, and it seemed to sound reasonable.

As shown in **Figure 1**, the MainPage.xaml.cs file defines several constant values and computes two integers that govern how the pixels of the display correspond to notes.

The range is from the C below middle C (a frequency of about 130.8Hz) to the C three octaves above middle C, about 2,093Hz. Two methods calculate a frequency and a relative amplitude (ranging from 0 to 1) based on the coordinates of a touch point obtained from the Touch.FrameReported event.

If you just use these values to control a sine wave oscillator, it won't sound like a theremin at all. As you move your finger across the screen, the program doesn't get an event for every single pixel along the way. Instead of a smooth frequency glide, you'll hear very discrete steps. To solve this problem, I created a special oscillator class, shown in **Figure 2**. This oscillator inherits a Frequency property but defines three more properties: Amplitude, DestinationAmplitude and DestinationFrequency. Using multiplicative factors, the oscillator itself provides gliding. The code can't actually anticipate how fast a finger is moving, but in most cases it seems to work OK.

Figure 3 shows the handler for the Touch.FrameReported event in the MainPage class. When a finger first touches the screen, Amplitude is set to a minimum value so the sound rises in volume. When the finger is released, the sound fades out.

As you can see from the code, the Theremin program generates just a single tone and ignores multiple fingers.

Although the theremin frequency varies continuously, the screen nevertheless displays lines to indicate discrete notes. These lines are colored red for C and blue for F (the colors used for harp strings), white for naturals and gray for accidentals (the sharps). After playing with the program awhile, I decided it needed some visual feedback indicating what note the finger was actually positioned on, so I made the lines widen based on their distance from the touch point. **Figure 4** shows the display when the finger is between C and C# but closer to C.

Latency and Distortion

One big problem with software-based music synthesis is latency—the delay between user input and the subsequent change in sound. This is pretty much unavoidable: Audio streaming in Silverlight requires that an application derive from MediaStreamSource and override the GetSampleAsync method, which supplies audio data on demand through a MemoryStream object. Internally, this audio data is maintained in a buffer. The existence of this buffer helps ensure that the sound is played back without any disconcerting gaps, but of course playback of the buffer will always trail behind the filling of the buffer.

Fortunately, MediaStreamSource defines a property named AudioBufferLength that indicates the size of the buffer in milliseconds of sound. (This property is protected and can be set only within the MediaStreamSource derivative prior to opening the media.) The default value is 1,000 (or 1 second) but you can set it as low as 15. A lower setting increases the interaction between the OS and the MediaStreamSource derivative and might result in gaps in the sound. However, I found that the minimum setting of 15 seemed to be satisfactory.

Another potential problem is simply not being able to crank out the data. Your program needs to generate tens or hundreds of thousands of bytes per second, and if it can't do this in an efficient manner, the sound will start breaking up and you'll hear a lot of crackling.

There are a couple ways to fix this: You can make your audio-generation pipeline more efficient (as I'll discuss shortly) or you can reduce the sampling rate. I found that the CD sampling rate of 44,100 was too much for my programs, and I took it down to 22,050. Reducing it further to 11,025 might also be necessary. It's always good to test your audio programs on a couple different Windows Phone devices. In a commercial product, you'll probably want to give the user an option of reducing the sampling rate.

Multiple Oscillators

The Mixer component of the synthesizer library has the job of assembling multiple inputs into composite left and right channels. This is a fairly straightforward job, but keep in mind that each input is a waveform with a 16-bit amplitude, and the output is also

Figure 6 **The Integer Version of SawtoothOscillator**

```
public class SawtoothOscillator : IMonoSampleProvider
{
  int sampleRate;
  uint angle;
  uint angleIncrement;

  public SawtoothOscillator(int sampleRate)
  {
    this.sampleRate = sampleRate;
  }

  public double Frequency
  {
    set
    {
      angleIncrement = (uint)(UInt32.MaxValue * value / sampleRate);
    }
    get
    {
      return (double)angleIncrement * sampleRate / UInt32.MaxValue;
    }
  }

  public short GetNextSample()
  {
    angle += angleIncrement;
    return (short)((angle >> 16) + short.MinValue);
  }
}
```

a waveform with a 16-bit amplitude, so the inputs must be attenuated based on how many there are. For example, if the Mixer component has 10 inputs, each input must be attenuated to one-tenth of its original value.

This has a profound implication: Mixer inputs can't be added or removed while music is playing without increasing or decreasing the volume of the remaining inputs. If you want a program that can potentially play 25 different sounds at once, you'll need 25 constant mixer inputs.

This is the case with the Harp application in the MusicalInstruments solution. I envisioned an instrument with strings that I could pluck with my fingertip, but which I could also strum for the common harp glissando sound.

As you can see from **Figure 5**, visually it's very similar to the theremin, but with only two octaves rather than four. The strings for the accidentals (the sharps) are positioned at the top, while the naturals are at the bottom, which somewhat mimics the type of harp known as the cross-strung harp. You can perform a pentatonic glissando (at the top), a chromatic glissando (in the middle) or a diatonic glissando (on the bottom).

For the actual sounds, I used 25 instances of a SawtoothOscillator class, which generates a simple sawtooth waveform that grossly approximates a string sound. It was also necessary to make a rudimentary envelope generator. In real life, musical sounds don't start and stop instantaneously. The sound takes a while to get going, and then might fade out by itself (such as with a piano or harp), or might fade out after the musician stops playing it. An envelope generator controls these changes. I didn't need anything as sophisticated as a full-blown attack-decay-sustain-release (ADSR) envelope, so I created a simpler AttackDecayEnvelope class. (In real life, a sound's timbre—governed by its harmonic components—also changes over the duration of a single tone, so the timbre should be controlled by an envelope generator as well.)

For visual feedback, I decided I wanted the strings to vibrate. Each string is actually a quadratic Bezier segment, with the central control point collinear with the two end points. By applying a repeating PointAnimation to the control point, I could make the strings vibrate.

In practice, this was a disaster. The vibrations looked great but the sound degenerated into extreme crackling ugliness. I switched to something a little less severe: I used a DispatcherTimer and offset the points manually at a much slower rate than a real animation.

After playing around with the Harp program a little while, I was unhappy with the flicking gesture required for plucking the strings, so I added some code to trigger the sound with just a tap. At that point, I probably should have changed the name of the program from Harp to HammeredDulcimer, but I let it go.

Avoiding Floating Point

On the Windows Phone device that I was using for most of my development, the Harp worked fine. On another Windows Phone it was extremely crackly, indicating that the buffers could not be filled quickly enough. This analysis was confirmed by halving the sampling rate. The crackling stopped with a sampling rate of 11,025Hz, but I wasn't ready to sacrifice the sound quality.

Instead, I started taking a close look at the pipeline that provided these thousands of samples per second. These classes—Mixer, MixerInput, SawtoothOscillator and AttackDecayEnvelope—all had one thing in common: They all used floating-point arithmetic in some way to compute these samples. Could switching to integer calculations help speed up this pipeline enough to make a difference?

I rewrote my AttackDecayEnvelope class to use integer arithmetic, and did the same thing with the SawtoothOscillator, which is shown in **Figure 6**. These changes improved performance significantly.

In the oscillators that use floating point, the angle and angleIncrement variables are of type double where angle ranges from 0 to 2π and angleIncrement is calculated like so:

$$\frac{2\pi \cdot Frequency}{Sample\ Rate}$$

For each sample, angle is increased by angleIncrement.

I didn't eliminate floating point entirely from SawtoothOscillator. The public Frequency property is still defined as a double, but that's only used when the oscillator's frequency is set. Both angle and angleIncrement are unsigned 32-bit integers. The full 32-bit values are used when angleIncrement increases the value of angle, but only the top 16 bits are used as a value for calculating a waveform.

Even with these changes, the program still doesn't run as well on what I now think of as my "slow phone" as compared to my "fast phone." Sweeping a finger across the whole screen still causes some crackling.

But what's true with any musical instrument is also true with electronic music instruments: You need to become familiar with the instrument, and know not only its power but its limitations. ■

CHARLES PETZOLD *is a longtime contributor to* MSDN Magazine. *His Web site is charlespetzold.com.*

THANKS to the following technical experts for reviewing this article: *Mark Hopkins*

Poetry of the Geeks

I was trying to explain neutrinos to a non-scientist the other day, in response to the faster-than-light claims being made (and since disproved). She wasn't grokking my explanations at all. I finally conveyed the way neutrinos pass through matter without interacting by quoting John Updike's poem, "Cosmic Gall" (bit.ly/fhFrQj):

At night, they enter at Nepal
and pierce the lover and his lass
From underneath the bed—you call
It wonderful; I call it crass

Sometimes only poetry can explain science, and technology, too. In Rudyard Kipling's poem, "McAndrew's Hymn" (bit.ly/htBITe), the Scottish chief engineer of a steamship cries aloud for such an aid to laypeople's understanding: "Lord, send a man like Robbie Burns to sing the Song o' Steam!" We geeks could benefit from that.

Being geeks, we mistakenly believe that we're normal; that our users resemble us. That produces terrible software, and not much better poetry. I doubled over with laughter at "The Girl Who Dreams in ADA" (bit.ly/yf1X22, credited to one Martin J. Carter). But I'll bet Antonio Carlos Jobim loses very little sleep worrying that it will replace his classic, "The Girl from Impanema," and not just because he's dead:

Fingers dance the Keyboard Samba
Like lines of code are goin' out of fashion
And each workstation
On compilation
Goes Argh

I doubt that my own composition, "I'm Just a Two-Bit Programmer on a 16-Bit Machine" (to the tune of "Stuck Inside of Mobile with Those Memphis Blues Again"), gives Bob Dylan many sleepless nights, either—even if I did get it published in the "Journal of Irreproducible Results."

We geeks do better writing haiku, perhaps because we respect intelligence most when it's exercised in the face of constraints. Here I salute the late Simba, who wrote the April Fools' edition of this column last year (msdn.microsoft.com/magazine/gg983482). You can see the Canadian influence, as I'd just returned from a gig in Toronto:

My cat pressed 'Reset'
Now she's a tennis racket
No more fur balls, eh?

The canonical haiku was written by Seth Schoen, to give the DeCSS DVD decryption algorithm the First Amendment protections due to artistic speech (bit.ly/fuhwA). This is just a tiny excerpt from his 456-triplet masterpiece, maintaining the 5-7-5 meter throughout:

Now help me, Muse, for
I wish to tell a piece of
controversial math,

for which the lawyers
of DVD CCA
don't forbear to sue:

that they alone should
know or have the right to teach
these skills and these rules.

I love reading Dr. Seuss' "Fox in Socks" to my daughters, even now, but I can't help adding a little technical spice. If Dr. Seuss had been a technical writer, he might have produced something like this (bit.ly/ySVKNU)—it's scary as hell how quickly my girls memorized it and can chant it:

If a packet hits a pocket on a socket on a port,
And the bus is interrupted as a very last resort.
And the address of the memory makes your floppy disk abort,
Then the socket packet pocket has an error to report!

Now that computer usage has become mainstream, mass-market artists have gotten into the act, or at least tried to. Jimmy Buffett seems to be still scratching his head with "Flesh and Bone" (bit.ly/xIjsN2):

I've got words but no processor
I've got feelings but I don't know DOS
so I just have to go back to BASICs
to try to get my point across

Don Middlebrook, leader of the bands Living Soul and the Pearl Divers (bit.ly/xg98n4), has a better handle on it. His song about online dating, "[You Can't] Download Love," asks: "What if he's a geek, and you click save instead of delete?" Tropical rocker Jeff Pike is up to the minute, with his breakaway ballad, "There's an App for That" (bit.ly/wxa95p).

I'll give the last word to David Pogue, technology critic at *The New York Times*. He also writes spoof songs, such as "The Bill Gates Song" (to the tune of "Chestnuts Roasting on an Open Fire"):

And so we're offering this simple prayer,
To Bill and all his MS grunts:
Since we all follow any standard you write,
Make it good, please,
Make it good, please,
Make it good, please, just once! ■

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.*