THE MICROSOFT JOURNAL FOR DEVELOPERS

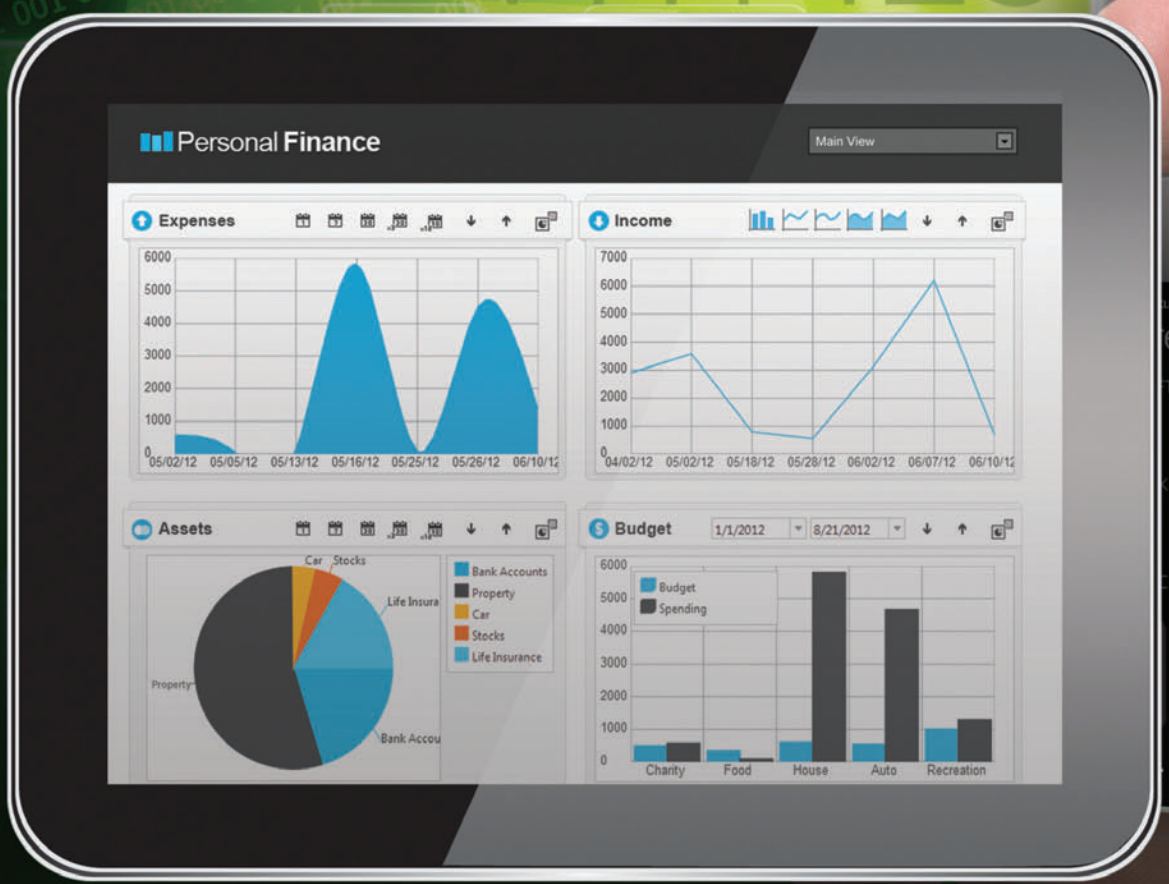
OCTOBER 2012 VOL 27 NO 10

msdn magazine

COLUMNS

Microsoft

Compatible with
Microsoft® Visual Studio® 2012



BRILLIANT UX

At Your Fingertips

infragistics.com/ **EXPERIENCE**

INFRAGISTICS™

DESIGN / DEVELOP / EXPERIENCE

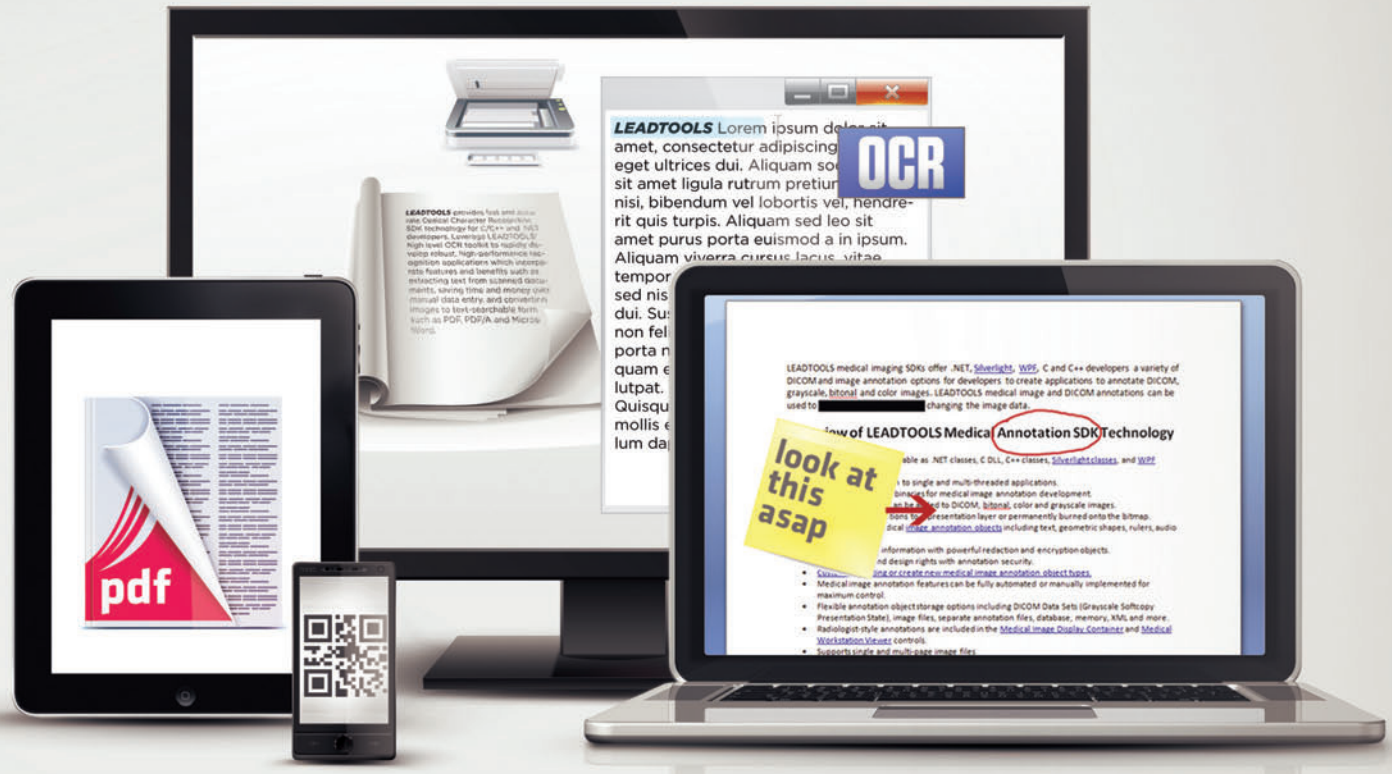
Infragistics Sales US 800 231 8588 • Europe +44 (0) 800 298 9055 • India +91 80 4151 8042 • APAC (+61) 3 9982 4545

msdn magazine

OCTOBER 2012 VOLUME 27 NUMBER 10

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.

LEADTOOLS®
DOCUMENT SDKs



OCR · BARCODE
PDF READ, WRITE, EXTRACT, VIEW & EDIT

DOCUMENT READERS & WRITERS · PREPROCESSING
FORMS RECOGNITION & PROCESSING · ANNOTATIONS

IMAGE FORMATS & COMPRESSION · SCANNING
VIRTUAL PRINTER

ZERO FOOTPRINT AND RICH CLIENT CONTROLS
FOR WEB, MOBILE & DESKTOP

C++ .NET WinRT HTML5
WEB SERVICES, WPF, ASP.NET & CLOUD

DOWNLOAD OUR 60 DAY EVALUATION
WWW.LEADTOOLS.COM

LEAD TECHNOLOGIES INCORPORATED

SAL ES@LEADTOOLS.COM
800.637.1840

Editor's Note

MICHAEL DESMOND

Cutting Edge

Dino Esposito has been at this job for a long time.

In fact, it was 1996 when the future Cutting Edge columnist began appearing in the pages of *Microsoft Internet Developer (MIND)* magazine, which in 2000 merged with *Microsoft Systems Journal (MSJ)* to become *MSDN Magazine*. As Esposito tells it, he contacted then-editor Josh Trupin at *MIND* about an article that had impressed him. During the course of the conversation, Esposito dropped the hint that he was anxious to write for the magazine.

"Josh swallowed the bait right away, and I then wrote my first article: 'Using CryptoAPI in MFC Applications.' It was a blast and I was asked to write a couple more," Esposito recalls.

More like a couple hundred more. After writing a number of features for *MIND*, Esposito wrote his first Cutting Edge column in January 1998 and never looked back. His contributions to the magazine continue in an unbroken string that spans Microsoft's remarkable Internet turnaround, the rise of the Microsoft .NET Framework and, most recently, the arrival of Windows 8 and the Windows Runtime. I don't have access to back issues of *MIND*, but I'm guessing Esposito has penned more than 150 Cutting Edge columns over the years.

"As the name suggests, the mission of the column was to show creative ways of doing things using both existing technologies and new technologies," Esposito says.

Esposito got his start working on the Microsoft platform—Windows 3.1 and Windows for Workgroups, specifically. "I've always been Microsoft-centric," Esposito says. "Not necessarily by choice; it just went that way. I liked it so much that I dedicated the lion's share of my time to write about Microsoft technologies. That was around the time the .NET Framework was released in the early 2000s."

The emergence of mobile development broadened Esposito's perspective. "I spent the past couple of years studying and working around a variety of different mobile projects," he explains, calling mobile "a key paradigm shift."

Fittingly, Esposito's column this month is the fourth in a five-part series on mobile site development. The current installment explores how to classify mobile devices to build Web sites that serve different markup to devices based on their specific capabilities. In October, Esposito will finish off the series with a look at the HTML5-based jQuery Mobile UI system for mobile device platforms.

When I ask Esposito if there was a particularly memorable column that stood out to him over the years, he notes that his body of work in ASP.NET DataGrids made him something of an Internet celebrity.

"Someone, at some point, called me the 'DataGrid Whisperer,' after the movie 'The Horse Whisperer.' It was common for me to receive e-mails from readers asking more and more," Esposito recalls. "I believe the best article in this regard was the one showing how to reorder, via drag-and-drop, columns of a DataGrid and make the new order persistent."

> "As the name suggests, the mission of the column was to show creative ways of doing things using both existing technologies and new technologies."

It's been 16 years and counting, and Esposito has come a long way from his days as the DataGrid Whisperer.

"Today I'm no longer the guy who spends the night downloading the latest bits and nightly builds. I only install RTM software and I'm only now, for example, looking into Windows 8. At the same time, the lesson I think I've learned is: Be able to know where you can find the tools that can help you do your job, and don't use tools you don't really need," he says.

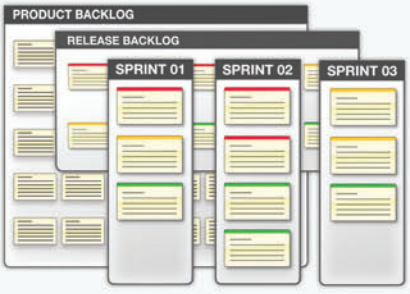
"This probably makes me much less geeky, but probably wiser," Esposito says. "Who knows?"

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

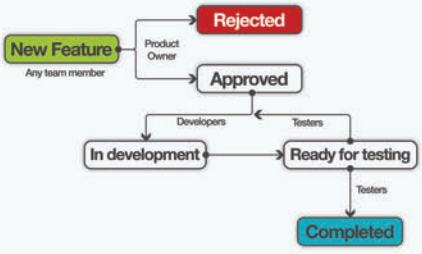
OnTime Scrum

Agile project management & bug tracking software

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



Easily manage product backlogs



Automate your workflow process

Project visibility with burndowns

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

$10 per month for up to 10 users

special small-team pricing

$7 per user per month

for teams of 11+ users

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

Visit **OnTimeNow.com/MSDN**

OnTime Scrum offers your dev team:

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

axosoft

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

DINO ESPOSITO

Mobile Site Development, Part 5: jQuery Mobile

In this column I'll share some thoughts about jQuery Mobile, an increasingly popular mobile technology.

As more and more Web sites are flanked by specialized sites to serve mobile device users, an equation takes form in the minds of many developers and managers. Simply put, it sounds like the following: If you love jQuery, then you're going to love jQuery Mobile, too. Fact is, a lot of people (including me) love jQuery, so potentially a huge number of developers would be inclined to use jQuery Mobile as the default choice for any mobile Web development without any further thinking.

I admit I followed this pattern exactly when I started mobile development. Sometimes it worked; sometimes it didn't. And, perhaps more important, sometimes I thought it worked, but in the end I didn't really get what my customers were looking for.

> Technology shouldn't dictate how you achieve your business needs; technology should only be a means to implement your immediate development goals.

Mobile development is not simply a branch of Web development. For the most part, it doesn't succeed by adapting (well, mostly simplifying) what you already have, whether that be use cases, layout, graphics, scripts or other related technologies. Well-done and cross-device mobile development is a true paradigm shift where the UX comes first and new types of professionals show up: information architects, interaction designers and content strategists. Mobile software is often designed and approved through mockups and then simply implemented, showing data at a given position and in a given format.

But note that technology shouldn't dictate how you achieve your business needs; technology should only be a means to implement your immediate development goals. It may sound like a fairly obvious, foregone statement, but—let's be honest—how many times have you seen this rule violated? Have you ever heard about the "Pet Technology" anti-pattern? You can read about it at bit.ly/OvJdx0. Sometimes it happens that a given technology is picked up for no compelling or business-oriented reason. Likewise, sometimes it happens that a given technology is picked up for the wrong task—or, perhaps worse yet, its use leads to a less-than-ideal approach for a given task.

In a today's rapidly changing mobile world, putting any technology ahead of UX and strategic considerations is a huge risk. In mobile development, use cases and vision come first, and technology is only a means—as it should always be!

Now I'll focus on the main subject of this column, jQuery Mobile, whose strong and weak points are often measured on a scale that counts what you *can* do instead of what you *want* to do. You can read more about jQuery Mobile at jquerymobile.org.

jQuery Mobile at a Glance

Overall, the most appropriate definition for jQuery Mobile comes from its Web site. It explains that jQuery Mobile is an HTML5-based UI system for all popular mobile device platforms. Furthermore, it says the jQuery Mobile code is built with "progressive enhancement" in mind, and results should have an easily "theme-able" design. In addition, jQuery Mobile is built on top of the rock-solid foundation of jQuery and jQuery UI. However, many professional developers assume that jQuery Mobile is a lightweight version of jQuery optimized for the constraints of typical mobile devices.

In truth, what's on the homepage of the jQuery Mobile site is precisely the true essence of the library. jQuery Mobile is a framework that's easy to use for many developers—or, at least, it turns out to be far easier to use than most of the other frameworks out there for building mobile solutions. It's a framework that helps with the UI of views that work well when displayed on mobile and touch-based devices. Not only are touch-based (and subsequently mobile-oriented) views easy to create, but they're also easy to tie together to form a navigation system with transitions and effects. Views created leveraging the facilities of jQuery Mobile work well with smartphones and to some extent also with older devices, leading to the key question: Is this really enough for your mobile Web development?

Where's the Mobile in jQuery Mobile?

Personally, I've used jQuery Mobile for building the UI of a Web-based application aimed at laptop or tablet users—no phones whatsoever!—and it worked beautifully. It was at this time that I first envisioned some sort of close relationship between jQuery UI and jQuery Mobile.

As surprising as it may sound, let's raise the point: Is jQuery Mobile really a framework for mobile development?

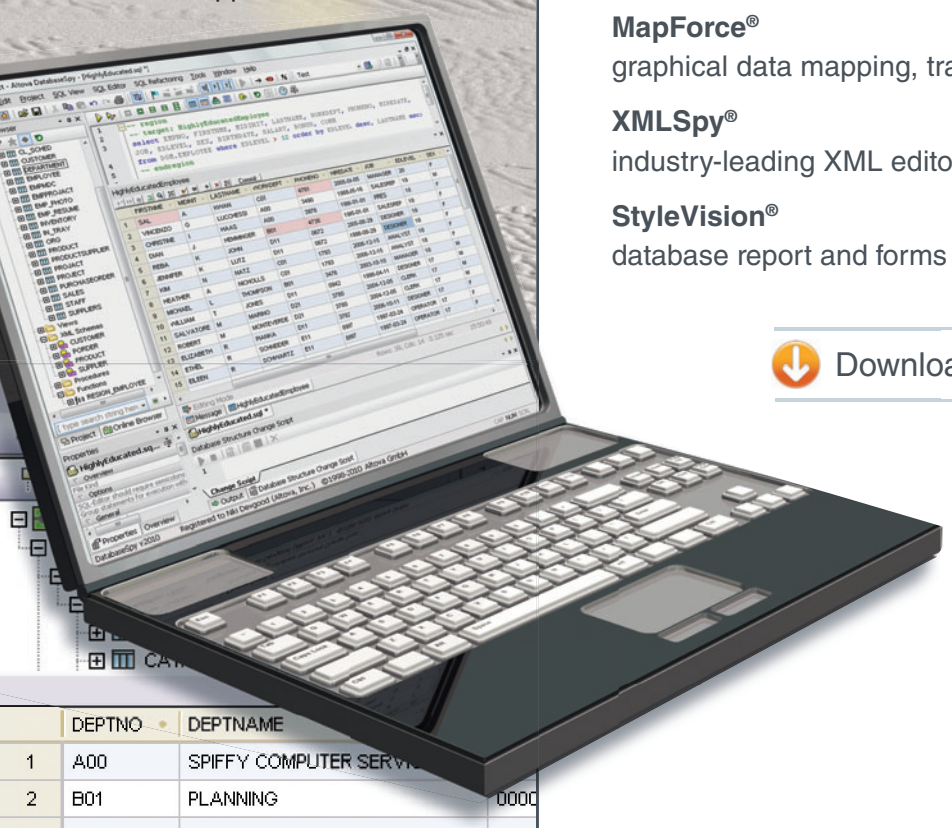
Deliver database results with the complete set of tools from Altova®

ALTOVA® missionkit®

Altova MissionKit® is a software development suite of industrial-strength XML, SQL, and data integration tools with powerful support for working with all major relational databases.

NEW in Version 2013:

- Support for SQL stored procedures in data mapping projects as a data source, target, or processing function
- Support for conditions on table columns and rows during report generation
- Smart Fix validation for automatic correction of XML validation errors
- Seamless integration options in Java applications



MissionKit database tools support all of the following:

- Microsoft® SQL Server®
- Oracle®
- IBM DB2®
- Sybase®
- MySQL®
- Postgre SQL
- Microsoft Access™
- Cloud database systems

Altova MissionKit includes multiple tools for working with databases:

DatabaseSpy®
SQL editor + database query, design & comparison tool

MapForce®
graphical data mapping, transformation & conversion tool

XMLSpy®
industry-leading XML editor with strong database integration

StyleVision®
database report and forms design tool

Download a 30 day free trial!

Try before you buy with a free, fully functional, 30-day trial from **www.altova.com**.

Scan to learn more about MissionKit database tools.

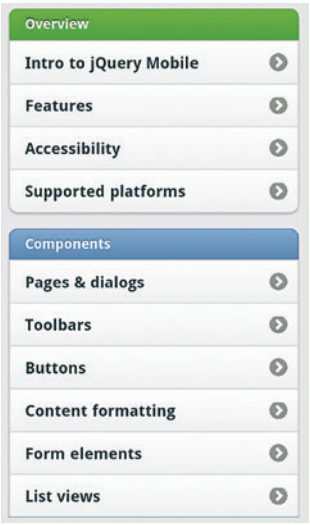


Figure 1 **A Sample Page on an A-Grade Android Device**

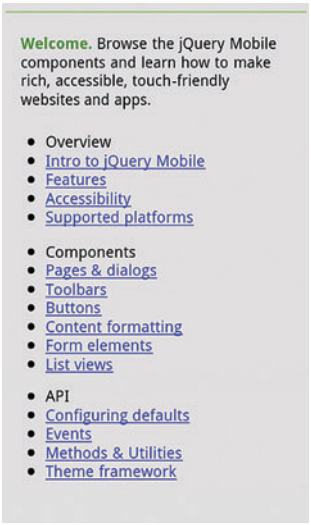


Figure 2 **A Sample Page on a C-Grade Android Device**

Based on the description of the technology on jquerymobile.org, you should consider jQuery Mobile as a primary tool for the UI of touch-based front ends. A touch-based front end is likely a mobile device (that is, tablet or smartphone), but mobile Web development has a lot more facets than just the front end.

Just using jQuery Mobile for arranging some UI that fits nicely in the small real estate of a device screen may be an approach that works in simple scenarios. If taken as an overall, comprehensive approach, however, it may be a bit simplistic.

Let's review what you may find in a mobile solution that goes beyond the capabilities of jQuery Mobile. First and foremost, you should have a clear vision of the site you're building, expressed through a great selection of use cases, a well-defined data flow between server and client, and, more important, a list of the mobile profiles you intend to serve. A mobile profile is a generic term to indicate the different families of devices your site intends to serve: smartphones, tablets, laptops, smart TVs and similar devices built in the past few years or so, or any cell phones that can connect to the Internet.

Not every mobile site needs to support multiple device profiles, even though this necessity is becoming increasingly stricter and more urgent. Supporting multiple device profiles means dealing with different views and dynamically adjusting the target response based on the device characteristics and capabilities. In my last column (msdn.microsoft.com/magazine/jj618291) I presented an approach to multi-serving based on a server-side device description repository such as Wireless Universal Resource File, or WURFL (see wurfl.sourceforge.net). What kind of out-of-the-box support do you get from jQuery Mobile in this regard?

jQuery Mobile and Responsive Design

Responsive design is the framework's ability to provide a made-to-measure version of a page for specific devices. Earlier in this article, quoting the jquerymobile.org site, I mentioned progressive enhancement as one of the pillars of the jQuery Mobile framework. Progressive enhancement is a bottom-up Web design pattern with which you create pages progressively enhancing a core that works on any browser. Progressive enhancement is the opposite of "graceful degradation," which opts for a top-down approach: Define the best experience and degrade as gracefully as you can if the browser misses some required capabilities. Progressive enhancement can be considered a pattern for making your Web experience more responsive.

Note that making a Web experience more responsive may include offering some ad hoc mobile support, but being responsive on the browser doesn't necessarily mean focusing on mobile clients.

So by using jQuery Mobile, you may not be worried about detecting device capabilities, as the library guarantees that the output would also work on down-level browsers. Let's review the core of the jQuery Mobile implementation of the progressive enhancement pattern.

Browser Partitioning

The jQuery Mobile library supports three browser profiles—named A, B and C—where A-grade browsers receive the most enhanced markup the library can serve and C-grade browsers just receive plain HTML content with no extra features, such as styles and AJAX. Each profile is characterized by a list of capabilities that browsers provide with respect to the library's needs. For example, support for CSS media queries is a key capability in jQuery Mobile, and it's a fundamental requisite for a browser to fall into the A group. In addition to CSS media queries, A-grade browsers support JavaScript, AJAX and full Document Object Model (DOM) manipulation. A-grade browsers are where the library operates at its fullest, and the list of A-grade browsers is updated frequently.

The basic difference between A-grade and B-grade browsers is the lack of support for AJAX. On B-grade browsers, jQuery Mobile stops using AJAX for page transitions and requests. So you might expect to be able to carry out operations successfully, but with a less-pleasant experience.

Finally, C-grade browsers are mostly legacy browsers with no support for media queries and limited support for JavaScript, CSS and DOM manipulation. On C-grade browsers, no standard DOM manipulation is applied to page elements, and plain HTML is served. The most updated matrix of browsers and grades is available at jquerymobile.com/gbs. **Figure 1** and **Figure 2** show the rendering of the same page on A-grade and C-grade browsers.

Reasonably, the jQuery Mobile library is not optimized for B-grade and C-grade browsers. As **Figure 2** shows, your users may still receive a good treatment, but, more important, you have no control over that. The HTML in **Figure 1** and **Figure 2** is fairly

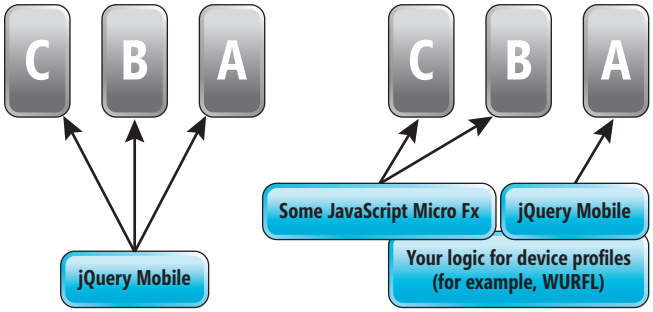


Figure 3 **Targeting Different Grades of Browsers**

simple and doesn't contain more than an unordered list. What if you have a much more sophisticated page with images, pop-ups, text blocks of various length and format, and buttons?

jQuery Mobile owes its nice graphics (as shown in **Figure 1**) to intensive DOM manipulation and AJAX features. Outside the realm of A-grade browsers, jQuery Mobile only guarantees that a page is viewable. You may find more abrupt transitions (no AJAX) and simpler DOM manipulation, or there may be a complete fallback to plain HTML. Regardless of the fact that jQuery Mobile can serve up to three distinct views of the same page, the basic HTML remains the same and the transformation rules are fixed and out of your control.

The bottom line is that if you need to support more than A-grade browsers, you'd probably be better off dropping jQuery Mobile browser grading and introducing your own logic to handle browser profiles and ad hoc views (see **Figure 3**).

The sample code in last month's column provides an example of this approach.

To summarize, jQuery Mobile is essentially a UI framework that does a great job of letting you use familiar HTML elements when you author a view. Next, it transforms plain HTML elements into semantically equivalent objects that work well in a touch-based environment and fit well on small screen sizes. jQuery Mobile implements the progressive enhancement pattern and doesn't leave any browser behind. However, this doesn't mean that by simply adopting jQuery Mobile you can effectively address the device-fragmentation problem—the large amount of significantly different mobile browsers and devices. Sometimes you're required to offer completely different views to different class profiles and even ensure that you partition browsers according to different logic. jQuery Mobile embraces the logic of responsive Web design and focuses on CSS media queries. As explained in my July 2012 column (msdn.microsoft.com/magazine/jj190798), CSS media queries are great to use with rich devices (tablets, laptops, smart TVs and perhaps smartphones) where all you want to do is reposition and hide elements that you can't display. Media queries require rich browsers and are not a mobile-specific feature.

I suggest that before you embrace jQuery Mobile, you ask the following question: Is it OK to designate certain smartphones (or analogous devices) as the entry point for really enjoying your site? If yes, then jQuery Mobile is more than OK for you. Otherwise, take it further and explore other options. ■

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

Thanks to the following technical expert for reviewing this article: *Christopher Bennage*

KENNY KERR

Back to the Future with Resumable Functions

I concluded my last column (msdn.microsoft.com/magazine/jj618294) by highlighting some possible improvements to C++11 futures and promises that would transform them from being largely academic and simplistic to practical and useful for the construction of efficient and composable asynchronous systems. In large part, this was inspired by the work of Niklas Gustafsson and Artur Laksberg from the Visual C++ team.

As a representation of the future of futures, I gave an example along these lines:

```
int main()
{
  uint8 b[1024];

  auto f = storage_read(b, sizeof(b), 0).then([&]()
  {
    return storage_write(b, sizeof(b), 1024);
  });

  f.wait();
}
```

Both the storage_read and storage_write functions return a future representing the respective I/O operations that might complete at some point in the future. These functions model some storage subsystem with 1KB pages. The program as a whole reads the first page from storage to the buffer and then copies it back to the second page of storage. The novelty of this example is in the use of the hypothetical "then" method added to the future class, allowing the read and write operations to be composed into a single logical I/O operation, which can then be seamlessly waited upon.

This is a huge improvement over the world of stack ripping that I described in my last column, yet in itself is still not quite the utopian dream of a coroutine-like facility supported by the language that I described in my August 2012 column, "Lightweight Cooperative Multitasking with C++" (msdn.microsoft.com/magazine/jj553509). In that column I successfully demonstrated how such a facility can be achieved with some dramatic macro trickery—but not without significant drawbacks, primarily related to the inability to use local variables. This month I want to share some thoughts on how this might be achieved in the C++ language itself.

I necessarily began this series of articles exploring alternative techniques for achieving concurrency with a practical solution because the reality is that we need solutions that work today. We do, however, need to look to the future and push the C++ community forward by demanding greater support for writing I/O-intensive applications in a more natural and productive manner. Surely writing highly scalable systems shouldn't be the exclusive purview of JavaScript and C# programmers and the rare C++ programmer with enough willpower. Also, keep in mind that this isn't just about convenience and elegance in programming syntax and style. The ability to have multiple I/O requests active at any given time has the potential to improve performance dramatically. Storage and network drivers are designed to scale well as more I/O requests are in flight. In the case of storage drivers, the requests can be combined to improve hardware buffering and reduce seek times. In the case of network drivers, more requests mean larger network packets, optimized sliding window operations and more.

Surely writing highly scalable systems shouldn't be the exclusive purview of JavaScript and C# programmers and the rare C++ programmer with enough willpower.

I'm going to switch gears slightly to illustrate how quickly complexity rears its ugly head. Rather than simply reading and writing to and from a storage device, how about serving up a file's contents over a network connection? As before, I'm going to start with a synchronous approach and work from there. Computers might be fundamentally asynchronous, but we mere mortals certainly are not. I don't know about you, but I've never been much of a multitasker. Consider the following classes:

```
class file { uint32 read(void * b, uint32 s); };
class net { void write(void * b, uint32 s); };
```

Use your imagination to fill out the rest. I just need a file class that allows a certain number of bytes to be read from some file. I'll further assume that the file object will keep track of the offset. Similarly, the net class might model a TCP stream where the data offset is handled by TCP via its sliding window implementation that's necessarily hidden from the caller. For a variety of reasons, perhaps related to caching or contention, the file read method might not always return the number of bytes actually requested.

All-in-One with dotConnect

ADO.NET Providers for Oracle, MySQL, PostgreSQL, SQLite and **Salesforce Now!**

Entity Framework

Fast, configurable EF provider with support for newest EF features

SQL Server BIS

BIS Data source for data export/import

Entity Developer

Powerful ORM model designer for Entity Framework and LinqConnect

Database Specific Extensions

Script, Loader, Alerter, Dump, Change Notification

dbMonitor

Monitor all the database interaction with free dbMonitor tool

NHibernate

Support for the well-known open-source ORM

ASP.NET DataSource component

SqlDataSource analog for different databases

Enterprise Library

Provider for Enterprise Library Data Access Block

ADO.NET Provider

Fast and advanced ADO.NET provider with direct data access, rich design-time

Workflow Foundation

Workflow Instance Store and Workflow Tracking support

ASP.NET Providers

ASP.NET Provider for storing state in different databases

LinqConnect

Fast and lightweight ORM solution, compatible with LINQ to SQL

devart

www.devart.com

It will, however, only return zero when the end of the file has been reached. The net write method is simpler because the TCP implementation, by design, thankfully does a huge amount of work to keep this simple for the caller. This is a basic imaginary scenario but pretty representative of OS I/O. I can now write the following simple program:

```
int main()
{
  file f = ...; net n = ...; uint8 b[4096];

  while (auto actual = f.read(b, sizeof(b)))
  {
    n.write(b, actual);
  }
}
```

Given a 10KB file, you can imagine the following sequence of events before the loop is exhausted:

```
read 4096 bytes -> write 4096 bytes ->
read 4096 bytes -> write 4096 bytes ->
read 2048 bytes -> write 2048 bytes ->
read 0 bytes
```

Like the synchronous example in my last column, it's not hard to figure out what's going on here, thanks to the sequential nature of C++. Making the switch to asynchronous composition is a bit more difficult. The first step is to transform the file and net classes to return futures:

```
class file { future<uint32> read(void * b, uint32 s); };
class net { future<void> write(void * b, uint32 s); };
```

That was the easy part. Rewriting the main function to take advantage of any asynchrony in these methods presents a few challenges. It's no longer enough to use the future's hypothetical "then" method, because I'm no longer simply dealing with sequential composition. Yes, it's true that a write follows a read, but only if the read actually reads something. To complicate matters further, a read also follows a write in all cases. You might be tempted to think in terms of closures, but that concept covers the composition of state and behavior and not the composition of behavior with other behavior.

I could start by creating closures only for the read and write operations:

```
auto read = [&]() { return f.read(b, sizeof(b)); };
auto write = [&](uint32 actual) { n.write(b, actual); };
```

Of course, this doesn't quite work because the future's then method doesn't know what to pass to the write function:

```
read().then(write);
```

To address this, I need some kind of convention that will allow futures to forward state. An obvious choice (perhaps) is to forward the future itself. The then method will then expect an expression taking a future parameter of the appropriate type, allowing me to write this:

```
auto read = [&]() { return f.read(b, sizeof(b)); };
auto write = [&](future<uint32> previous) { n.write(b, previous.get()); };

read().then(write);
```

> My hope is that the C++ community will embrace resumable functions, or something like them.

This works, and I might even want to improve composability further by defining that the expression the then method expects should also return a future. However, the problem remains of how to express the conditional loop. Ultimately, it proves to be simpler to reconsider the original loop as a do...while loop instead, because this is easier to express in an iterative way. I could then devise a do_while algorithm to mimic this pattern in an asynchronous manner, conditionally chaining futures and bringing the iterative composition to an end based on the result of a future<bool> value, for example:

```
future<void> do_while(function<future<bool>()> body)
{
  auto done = make_shared<promise<void>>();
  iteration(body, done);
  return done->get_future();
}
```

The do_while function first creates a reference-counted promise whose ultimate future signals the termination of the loop. This is passed to the iteration function along with the function representing the body of the loop:

```
void iteration(function<future<bool>()> body, shared_ptr<promise<void>> done)
{
  body().then([=](future<bool> previous)
  {
    if (previous.get()) { iteration(body, done); }
    else { done->set_value(); }
  });
}
```

This iteration function is the heart of the do_while algorithm, providing the chaining from one invocation to the next, as well as the ability to break out and signal completion. Although it might look recursive, remember that the whole point is to separate the asynchronous operations from the stack, and thus the loop doesn't actually grow the stack. Using the do_while algorithm is relatively easy, and I can now write the program shown in **Figure 1**.

The do_while function naturally returns a future, and in this case it's waited upon, but this could just as easily have been avoided by storing the main function's local variables on the heap with shared_ptrs. Inside the lambda expression passed to the do_while function, the read operation begins, followed by the write operation. To keep this example simple, I assume that write will return immediately if it's told to write zero bytes. When the write operation

Figure 1 **Using a do_while Algorithm**

```
int main()
{
  file f = ...; net n = ...; uint8 b[4096];

  auto loop = do_while([&]()
  {
    return f.read(b, sizeof(b)).then([&](future<uint32> previous)
    {
      return n.write(b, previous.get());

    }).then([&]()
    {
      promise<bool> p;
      p.set_value(!f.eof);
      return p.get_future();
    });
  });

  loop.wait();
}
```

completes, I check the file's end-of-file status and return a future providing the loop condition value. This ensures that the body of the loop will repeat until the file's content is exhausted.

Although this code isn't particularly obnoxious—and, indeed, is arguably a lot cleaner than stack ripping—a little support from the language would go a long way. Niklas Gustafsson has already proposed such a design and called it "resumable functions." Building on the improvements proposed for futures and promises and adding a little syntactic sugar, I could write a resumable function to encapsulate the surprisingly complex asynchronous operation as follows:

```
future<void> file_to_net(shared_ptr<file> f, shared_ptr<net> n) resumable
{
  uint8 b[4096];

  while (auto actual = await f->read(b, sizeof(b)))
  {
    await n->write(b, actual);
  }
}
```

The beauty of this design is that the code has a striking resemblance to the original synchronous version, and that's what I'm looking for, after all. Notice the "resumable" contextual keyword following the function's parameter list. This is analogous to the hypothetical "async" keyword I described in my August 2012 column. Unlike what I demonstrated in that column, however, this would be implemented by the compiler itself. Thus there would be no complications and limitations such as those I confronted with the macro implementation. You could use switch statements and local variables—and constructors and destructors would work as expected—but your functions would now be able to pause and resume in a manner similar to what I prototyped with macros. Not only that, but you'd be freed from the pitfalls of capturing local variables only to have them go out of scope, a common mistake when using lambda expressions. The compiler would take care of providing storage for local variables inside resumable functions on the heap.

In the earlier example you'll also notice the "await" keyword preceding the read and write method calls. This keyword defines a resumption point and expects an expression resulting in a future-like object that it can use to determine whether to pause and resume later or whether to simply continue execution if the asynchronous operation happened to complete synchronously. Obviously, to achieve the best performance, I need to handle the all-too-common scenario of asynchronous operations completing synchronously, perhaps due to caching or fast-fail scenarios.

Notice that I said the await keyword expects a future-like object. Strictly speaking, there's no reason it needs to be an actual future object. It only needs to provide the necessary behavior to support the detection of asynchronous completion and signaling. This is analogous to the way templates work today. This future-like object would need to support the then method I illustrated in my last column as well as the existing get method. To improve performance in cases where the result is immediately available, the proposed try_get and is_done methods would also be useful. Of course, the compiler can optimize based on the availability of such methods.

This isn't as far-fetched as it might seem. C# already has a nearly identical facility in the form of async methods, the moral equivalent of resumable functions. It even provides an await keyword that works in the same way as I've illustrated. My hope is that the C++ community will embrace resumable functions, or something like them, so that we'll all be able to write efficient and composable asynchronous systems naturally and easily.

For a detailed analysis of resumable functions, including a look at how they might be implemented, please read Niklas Gustafsson's paper, "Resumable Functions," at bit.ly/zvPr0a. ■

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

THANKS to the following technical expert for reviewing this article: *Artur Laksberg*

JOSEPH FULTZ

Windows Azure In-Role Caching

The old notion that luck favors the prepared is meant to convey the idea that no matter how lucky you are, you need to be prepared in order to capitalize on the lucky occurrence. I've often thought this statement describes caching pretty accurately. If you're lucky enough for the universe to align in such a way as to drive high use of your site and services, you'd better be prepared to serve the content quickly.

Back in January I covered some concepts related to caching that focused rather tactically on some coding approaches (msdn.microsoft.com/magazine/hh708748). With the addition of the dedicated and co-located roles for caching in the Windows Azure Caching (Preview), which I'll refer to here as simply Caching Preview, I felt it would be useful to consider how to use these roles as part of the overall solution architecture. This won't be an exhaustive coverage of caching features; instead, it's intended to be a designer's view of what to do with the big blocks.

Figure 1 **Remaining RAM**

Virtual Machine Size	Total RAM	10%/90% Reserved/Available	20%/80% Reserved/Available	40%/60% Reserved/Available
X-Small	768MB	N/A		
Small	1.75GB	175MB/1.575GB	350MB/1.4GB	700MB/1.05GB
Medium	3.5GB	350MB/3.15GB	700MB/2.8GB	1.4GB/2.1GB
Large	7GB	700MB/6.3GB	1.4GB/ 5.6GB	2.8GB/4.2GB
X-Large	14GB	1.4GB/12.6GB	2.8GB / 11.2GB	5.6GB/8.4GB

A Cache by Any Other Name ...

... is not the same. Sure, the back-end implementation is pretty similar and, like its forerunner Windows Azure Shared Caching, Caching Preview will move the data you fetch into the local cache client. More important, though, Caching Preview introduces some capabilities missing from the Shared Cache, so switching to role-based caching not only expands the available feature set, it also gives you better control over the deployment architecture. To start, let's clarify the primary difference between the dedicated and co-located roles: configuration.

When configuring the cache nodes, you have the option of dedicating the entire role to the cache or setting aside just a percentage of the role. Just as a way to quickly consider the implications of reserving RAM for the co-located cache, take a look at **Figure 1**, which shows remaining usable RAM after the cache reservation. (Note that the co-located option isn't available in the X-Small instance.)

Often the first thought is to simply choose some medium or small size and allocate some amount of memory. As long as the amount of memory allocated is sufficient for its intended use and within the boundary of the available RAM, this is a fine approach. However, if the number of objects is high and there's a reasonable expectation that the cache client on each machine might be holding its maximum number of objects, the result could be unexpected memory pressure. Moreover, too little cache RAM could lead to unwanted cache evictions, reducing the overall effectiveness of the cache.

Figure 2 shows the percentage of RAM use based on virtual machine (VM) size. The chart is based on the one at msdn.microsoft.com/library/hh914152, which shows the amount of RAM available for caching in dedicated mode.

In my co-located grid (**Figure 1**), I didn't go beyond 40 percent allocation for the co-located type because I assumed I'd need a majority of the RAM for the application. In comparison, the dedicated version usually provides more RAM, but appears to hit maximum efficiency of RAM allocation at the large VM size. In that sense, two medium VMs are less useful than one large. Of course, one large instance can't help with options such as high availability (HA), which duplicates your data, that you might want in your caching infrastructure. Still, it's worth weighing the needs of space against the need for redundancy and choosing a configuration that not only meets technical needs, but also optimizes cost.

When caching is done purposefully, a RAM drought typically isn't an issue. However, in cases where the shared cache is used to back session objects and/or the output cache, the situation is a bit more challenging because of the tendency to use session for everything and the difficulty in predicting exact load. For example, if you're running a Model-View-Controller app that has deep models you're placing in Session and you increase the maximum number of objects for the cache client, you might encounter undesired results under a medium or greater load. This would surface as slower site performance caused by evictions from the shared cache, from memory pressure you didn't expect; don't forget, the cache client is likely holding more RAM than anticipated due to the combination of an increased max object count and a deep graph. The framework helps you out a bit by compressing the serialized objects, but for such a precious and finite resource as RAM, diligence in accounting is the best practice, especially when trying to share the RAM among the application, output cache, session objects, data cache and cache client. To assist you in sizing your cache,

Figure 2 **Cache Use for Dedicated Role**

Virtual Machine Size	Available Memory for Caching	% of RAM Use based on Virtual Machine Size
Small	Approximately 1GB	57%
Medium	Approximately 2.5GB	57%
Large	Approximately 5.5GB	79%
X-Large	Approximately 11GB	79%

componentart.com/windows8

Microsoft has published the Capacity Planning Guide spreadsheet, which you can find at msdn.microsoft.com/library/hh914129.

Regions

Regions add some nice functionality, but at a cost. The cost is that the region is pinned to a single server forcing all requests for objects in the region to be bottlenecked through that cache host when it isn't stored in the cache client. The upside is that using regions provides tagging capability. My favorite use of regions is to hold pre-fetched reference data. This might seem folly at first, because of the bottleneck problem, but let's see.

To consider the cache use, let's postulate that I have a catalog of 10,000 products with up to eight variants for each product, meaning a catalog of potentially 80,000 items. If each object representing an item averages 1K, that's about 82MB to shuffle around on each request, as well as take up space in the cache client. In addition, there will be some number of virtual catalogs that are either a full copy or subset of the original, so I could end up with an explosion of reference data to be shuffled about, all served by the single region host (see **Figure 3**).

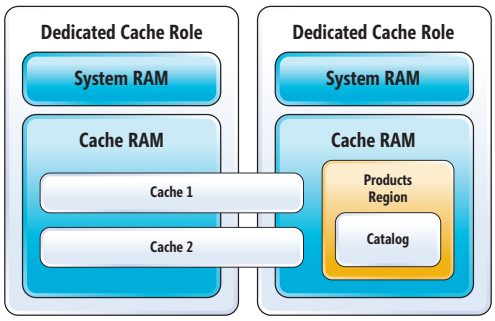


Figure 3 **Cache Layout with a Single Region**

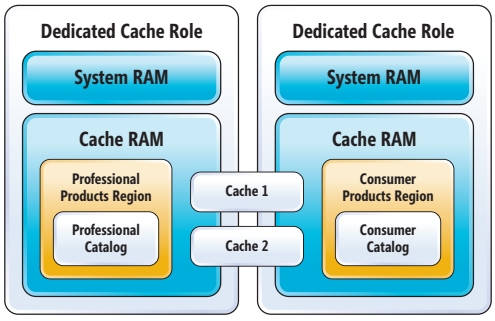


Figure 4 **Cache Layout with Two Regions**

However, with a little work I can create more regions to hold subsections of the data. For example, I might break my catalog into departments or segments. I could, for example, create one region for consumer products and one for professional products, resulting in something like what's shown in **Figure 4**.

This provides a little more granularity in the cache, enabling the use of smaller roles to hold all of my cache objects, ease cache updates, and decrease traffic by reducing the queries to each role and by filtering through the use of tag queries.

The ability to tag content is the primary function driving the use of regions. Thus, I can mark the content in my catalogs; for computers, for example, I might have tags such as: "laptop," "4GB," "8GB," "15 in.," "HD Audio," "Desktop" and so on. In this way I can enable such UI elements as a faceted product search for navigation by using a call to one of the GetObjectsByTag methods. It also means reengineering the data access layer and, in some regard, treating the cache more as the primary data source by which the queries on the facets (tags) of data are satisfied.

An interesting way to take advantage of this feature is to use Windows Azure Storage Tables as the back-end datastore, but to pre-fetch the data, tag it and put it into cache. This provides some of the filtering missing from the current incarnation of Storage Tables while keeping costs to a minimum.

Using regions provides a lot of flexibility in retrieving cached data, but do note the specific type of strain it places on the deployment infrastructure. Still, regions are handy as a means to pre-fetch and access reference data.

High Availability

There's a funny thing to consider with HA caches—you use an HA cache to be careful, but you need to be careful when using HA. At least, you need to be appropriately thoughtful about what really needs to be highly available.

Because every role enabled for duplication doubles the amount of space needed for the actual objects, you run out of RAM much faster. Thus, as a matter of design, it's best to use HA only for those features that actually need it or that would vastly improve UX so as to not arbitrarily drive up costs or artificially trigger cache evictions due to memory starvation resulting from overconsumption by duplicated caches.

I've seen some guidance that suggests putting session objects into the HA cache so you can query across users in active sessions based on certain tag values. In most instances, this would not be a useful approach as it inequitably distributes the load when retrieving session objects from cache; that load pattern should adhere more closely to the load balancing of the site. Furthermore, because you may well have a lot of empty anonymous profiles in addition to under-identified registered users, tag-based search for such entities as user profiles are actually more limiting than helpful.

I suggest you put user objects, sessions, output caching and the like into their own named caches, but don't enable them for duplication. In cases where edit data is tied to a session, you might consider backing up the session with an HA cache depending on where you are in the application's lifecycle. If the app is still being designed and

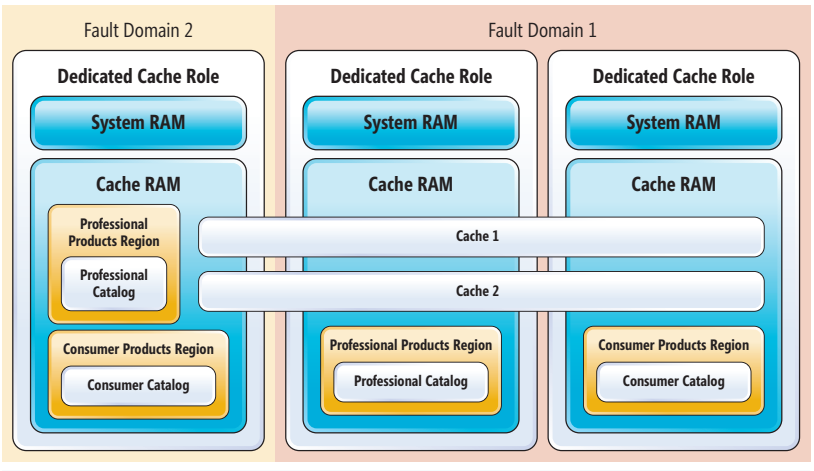


Figure 5 **High Availability and Fault Domains**

componentart.com/windows8

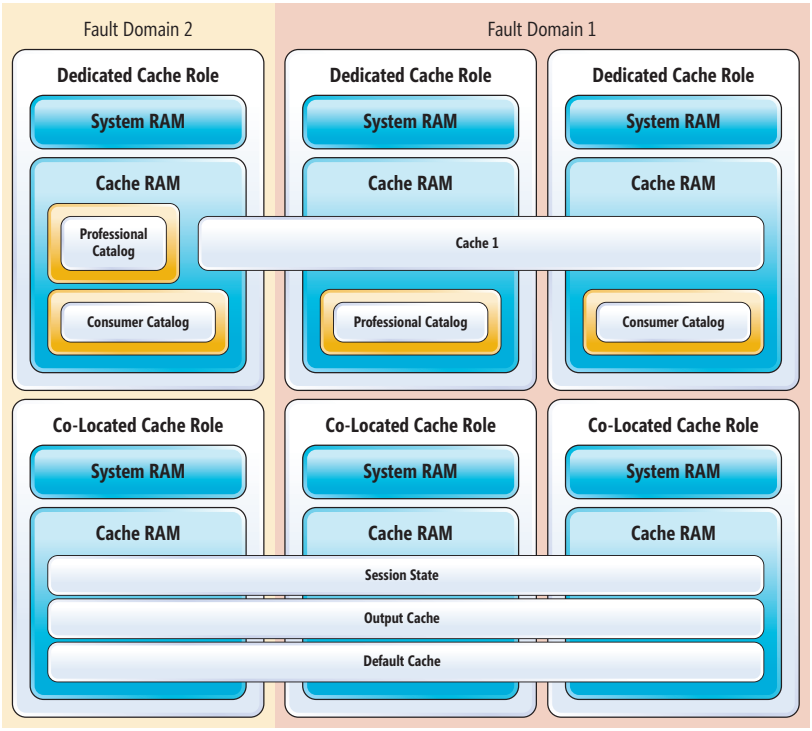


Figure 6 **Cache Deployment Possibilities**

created, it's better to separate those in-situ state objects and place them in an HA cache outside of the session. This lets you manage the edit data related to a user beyond the scope of the session and to keep the session in a much more evenly spread cache. However, if your app is further along and you have data you don't want to lose for legal, financial or just ease-of-use reasons that are bound up with the session object, it's acceptable to just wire a session to the HA cache—just be sure to specifically stress that in your load tests and know the limits of the implementation. Beyond data that's important due to its content or its point in a process—such as the data backing an editing interface—the types of data that are immediate targets for HA are large reference sets, pre-fetched data and pre-calculated data.

The commonality among all of these is the cost to pre-fetch that data again. In the case of pre-fetched or pre-calculated reference data, the start-up cost to prime the cache can be quite significant and losing the data during runtime might have a severe and even catastrophic impact on the site execution. **Figure 5** depicts how the cache objects might be allocated with HA turned on. Because the duplicates must be in a different fault domain, you can see how the duplicates reduce the overall RAM available for the cache. This is not to say it's always bad so much as it is to say it's what's necessary. I'm simply suggesting a conscious awareness of the potential impact of HA.

Figure 7 **Configuration Preferences**

Type of Data	Use HA	Use Region	Dedicated	Co-Located
Session				X
Output				X
General Data			X	X
Pre-fetch	X		X	
Pre-calc	X		X	
Important Data	X			
Filterable		X	X	

Lego House

Developers often like to think of development as building with Lego blocks; you create the basic blocks and snap them together into a useful application. This idea remains true even as we move further up the application stack from functions to objects to components to infrastructure. To that end, I want to leave you with some design guidance.

First, use all of the tools to your advantage. Don't settle on only HA or on no HA because one way is easier. Don't use only region-based caches because you can search them; or forego them because they get pinned to an instance. Rather, construct your caching infrastructure to suit your needs.

Figure 6 shows the dedicated roles I use to house my duplicated caches and the regions I use for more richly searchable caches. As a general rule, I favor dedicated cache roles for housing regions, because I don't want to load down a role that's serving user traffic with all of the traffic for cache fetches related to a given region. The bottom row in **Figure 6** depicts using the co-located style of caching to hold session, output and various other data I might cache during the execution of my application. This is not to say I'm stuck with dedicated or co-located roles as depicted, as that mostly depends on the RAM requirements of the items I intend to cache in the roles. Indeed, for many implementations, the bottom row alone will do the trick with no need for HA, regions or the large amount of RAM afforded by the dedicated role.

Finally, **Figure 7** is a grid that identifies my starting point for different types of data when I'm considering how to architect my cache deployment.

This is in no way meant to dictate usage or suggest a role or feature is useful only for the slot I've identified. It's just my starting point. If I had a large set of pre-fetched data I wanted to be able to search by tag, I'd combine the items I marked, ending up with a dedicated cache role that uses regions and HA. As an example of when I might deviate from my starting point, if I have a deployed application that uses session to cache its models while the user edits data, I would most likely toss my tendency to put session in a co-located cache and not only put it in a dedicated role, but enable HA as well.

So, if you're lucky enough to have a busy site, make sure your luck will continue by properly preparing your caching infrastructure. ■

Joseph Fultz *is a software architect at Hewlett-Packard Co., working as part of the HP.com Global IT group. Previously he was a software architect for Microsoft, working with its top-tier enterprise and ISV customers to define architecture and design solutions.*

Thanks to the following technical experts for reviewing this article: *Rohit Sharma and Hanz Zhang*

Horizontal Scalability for Parallel Execution of Tasks

Jesus Aguilar

The Task Parallel Library (TPL), introduced in the Microsoft .NET Framework 4, empowers application developers to create solutions that leverage the power of parallel processing in a multicore computer. In many scenarios, however, the ability to scale vertically (by adding more cores) is constrained by a number of factors, including cost and hosting limitations. In such cases, if scalability is required, it's desirable to distribute the processing across an array of servers; cloud hosting is an example of this. In this article I'll describe the key aspects (including an implementation) of a conceptual solution to accomplish this using many of the new features of the .NET Framework 4.5.

Basic Assumptions

The approach I'll describe requires several technologies beyond the TPL, which include:

- Task Parallel Library (TPL)
- Windows Communication Foundation (WCF)
- Managed Extensibility Framework (MEF)

This article discusses:

- The remote task client, task coordinator and the task execution nodes
- Tracking and handling client requests
- Queuing and throttling
- Sending requests to the task execution nodes and handling results
- Requesting and handling a cancellation

Technologies discussed:

Task Parallel Library, Microsoft .NET Framework 4.5, Windows Communication Foundation, Managed Extensibility Framework

Code download available at:

archive.msdn.microsoft.com/mag201210TPL

Note that I'll discuss these only in the context of the problem I'm trying to solve. I'm assuming you have a good understanding of these technologies.

Remote Task Client, Task Coordinator and Task Execution Nodes

The remote task client is the client-side layer that will hide the complexity resulting from the semantics of using a distributed environment. The remote task client interacts directly with the task coordinator, which then becomes the entry point to the underlying infrastructure. At a high level, the task coordinator has the following attributes:

1. It's the only point of contact with the clients.
2. It exposes the necessary services to request the execution of tasks on the scalable platform, as well as the cancellation of a given task.
3. It handles the throttling and queuing of the task execution requests, which supports the healthy operation of the environment.

The task execution nodes are the hosts of the processes in which the tasks will run. The actual implementations of the tasks that will be executed by the TPL reside in the task execution nodes.

Here are the key aspects of these logical layers and the flow of information:

1. The remote task client requests the execution of one or more tasks.
2. The task coordinator submits the request to the task execution nodes.
3. The task execution nodes execute the tasks and update the status of each request in the task coordinator.
4. The task coordinator updates the client with the results of the execution for each request.

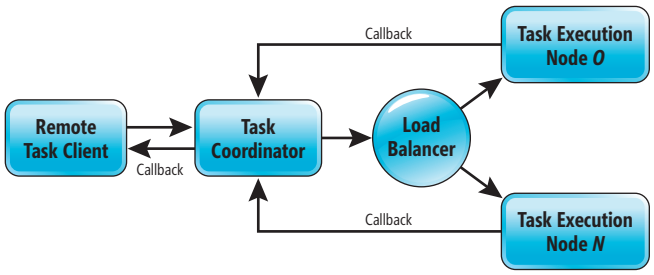


Figure 1 **Scaling Tasks Horizontally**

5. The task execution nodes reside behind a load balancer so more nodes can be added as needed, providing the ability to scale horizontally.

Figure 1 depicts the logical layers and the flow of information.

Note how the task execution nodes update the task coordinator, which then updates the remote task client. I'm going to describe an implementation based on bi-directional communication between the client and the task coordinator and between the task coordinator and the task execution nodes. In WCF terms, this implies the use of a duplex channel that allows the task execution nodes to call back the task coordinator and, subsequently, the task coordinator to do the same to update the client. I'll showcase the use of WebSockets to achieve this bi-directional communication approach. The WebSockets transport is implemented as a new binding in the .NET Framework 4.5 and is available for Windows 8. You'll find more information about the binding at bit.ly/SOLNiU.

The Client and the Task Coordinator

Now that you understand the three main logical layers—remote task client, task coordinator and task execution nodes—let's start by discussing the implementation of the remote task client. Note that when I use the term "client" throughout this article, I'm referring to the remote task client.

As I mentioned earlier, the value proposition of the client is the ability to hide the complexity of the underlying components. One way it achieves this is by providing an API that gives the impression of local execution of tasks, despite the fact they might be executing elsewhere. The code in **Figure 2** shows the public methods of the RemoteTaskClient class.

Figure 2 **Public Methods of the Class RemoteTaskClient**

```
public class RemoteTaskClient<TResult> : IDisposable
{

  public void AddRequest(string typeName, string[] parameters, CancellationToken tk)
  {...}

  public void AddRequest(string typeName, string[] parameters)
  {...}

  public Task<TResult>[] SubmitRequests()
  {...}


  public RemoteTaskClient(string taskCoodinatorEndpointAddress)
  {...}

  public void Dispose()
  {...}

}
```

You can use the AddRequest method to add requests for remote execution. For each request you need to specify the typeName (which is the type of the actual implementation containing the delegate that the infrastructure will run remotely as a TPL task) and the associated parameters. Then you can submit the requests via the SubmitRequest method. The result of submitting a request is an array of TPL tasks, one for each request. This approach will allow you to manage the resulting TPL tasks as you would if they were local. For example, you can submit various requests and wait for them to complete, like so:

```
using (var c = new RemoteTaskClient<int>("..."))
  {

    c.AddRequest("...", null);
    c.AddRequest("...", null);

    var ts = c.SubmitRequests();

    Task.WaitAll(ts);

    foreach (var t in ts)
      Console.WriteLine(t.Result);
  }
```

Before going into the details of the implementation of the RemoteTaskClient, let's look at the service operations and the data contracts that the task coordinator exposes. Understanding these contracts before reviewing the implementation of the RemoteTaskClient will give you additional context because the implementation of the client relies on these services.

The code in **Figure 3** shows the service operations the task coordinator exposes to the client. Through the SubmitRequest operation, the client has the ability to request the execution of one or more TPL tasks. The client can also request the cancellation of a particular TPL task that isn't complete, via the CancelTask operation. Note that the UpdateStatus operation is a callback. It's through a client-side implementation of this callback contract that the task coordinator will update the status at the client.

Let's look at the data contract that represents the task execution request. This is the data entity the client will send to the task coordinator, which in turn will submit the request to the task execution node where the actual execution will occur. The class STask, shown in **Figure 4**, models a task execution request. Using the properties STaskTypeName and STaskParameters, the client can set the type of task it wants to execute, with the relevant parameters. The task

Figure 3 **Service Operations**

```
[ServiceContract(CallbackContract = typeof(ITaskUpdateCallback))]
  public interface ITaskCoordinator
  {
    [OperationContract(IsOneWay = true)]
    void SubmitRequest(List<STask> stask);


    [OperationContract]
    bool CancelTask(string Id);

  }

  public interface ITaskUpdateCallback
  {
    [OperationContract (IsOneWay = true)]
    void UpdateStatus(string id, STaskStatus status, string result);

  }
```

coordinator will use the property Id as a unique identifier that the logical layers can use to correlate the request with the actual TPL task running in the system.

Now let's go back to the RemoteTaskClient and discuss how I'm planning to correlate the local TPL task with the result of the execution in the task execution nodes. The TPL has a convenient class, TaskCompletionSource<TResult>, I can use to create a TPL task and control its lifecycle. This mechanism lets me signal when a given task is completed, canceled or faulted. The implication here is that each request that goes to a task execution node (via the task coordinator) must be correlated to an instance of the TaskCompletionSource. For this, I implemented the class ClientRequestInfo, shown in **Figure 5**.

Figure 6 shows the implementation of the constructor of this class.

Notice I'm opening a duplex channel to the task coordinator and creating a callback instance of the type CallbackHandler. CallbackHandler receives as a parameter _requests, which contains instances of ClientRequestInfo. The rationale is that the _requests dictionary holds all the active instances of the client requests (and the instances of TaskCompletionSource that are associated with them), and the CallbackHandler will handle the updates from the task coordinator. Because multiple service requests will be updating the _requests dictionary, I need to guarantee thread-safety, hence the need for creating this as an instance of ConcurrentDictionary.

Figure 7 shows the implementation of the CallbackHandler class.

Next, let's look at the implementation of the AddRequest and SubmitRequest methods, as shown in **Figure 8**.

Figure 4 **The STask Class**

```
[DataContract]
  public class STask
  {
    [DataMember]
    public string Id
    { get; set; }

    [DataMember]
    public string STaskTypeName
    { get; set; }


    [DataMember]
    public string[] STaskParameters
    { get; set; }

  }
```

Figure 5 **The Class ClientRequestInfo**

```
internal class ClientRequestInfo<TResult>
{
  internal STask TaskExecutionRequest
  { get; set; }
  internal TaskCompletionSource<TResult> CompletionSource
  { get; set; }

  internal ClientRequestInfo(string typeName, string[] args)
  {
    TaskExecutionRequest = new STask()
      {Id = Guid.NewGuid().ToString(), STaskTypeName =typeName,
        STaskParameters = args };

    CompletionSource = new TaskCompletionSource<TResult>();
  }
}
```

Figure 6 **The ClientRequestInfo Constructor**

```
ITaskCoordinator _client;
ConcurrentDictionary<string, ClientRequestInfo<TResult>> _requests = new
ConcurrentDictionary<string, ClientRequestInfo<TResult>>();

  public RemoteTaskClient(string taskCoordinatorEndpointAddress)
  {
    var factory = new DuplexChannelFactory<ITaskCoordinator>
                  (new InstanceContext(new CallbackHandler<TResult>(_requests)),
                  new NetHttpBinding(),
                  new EndpointAddress(taskCoordinatorEndpointAddress));

    _client = factory.CreateChannel();

    ((IClientChannel)_client).Open();
  }
```

Tracking Client Requests

As you saw in the previous section, the client interacts solely with the task coordinator and it's the responsibility of the task coordinator to handle the requests from the client and subsequently update the client with the results of the execution of the TPL task. As with the client, this requires persisting the original request in some form. It also requires keeping track of the corresponding callback instance (which allows communication with the client); the channel to the task execution nodes associated with the connection (needed, as you'll see later, in cancellation scenarios); a unique identifier that groups all the task execution requests associated with a single call to a task execution node (to determine when the channel is no longer needed); as well as the status and result of the execution. **Figure 9** shows the definition of the STaskInfo class, the entity that will hold this information. In addition, I'll use a single instance of the ConcurrentDictionary<TKey,TValue> as the persistence mechanism.

Finally, note that _submissionTracker is contained in class CoordinatorContext. I will use this class to implement the main functionality of the task coordinator.

Figure 7 **The CallbackHandler Class**

```
[ServiceBehavior(ConcurrencyMode = ConcurrencyMode.Reentrant)]
public class CallbackHandler<TResult> : ITaskUpdateCallback
{
  ConcurrentDictionary<string, ClientRequestInfo<TResult>> _requests;

  public void UpdateStatus(string id, STaskStatus status, Object result)
    {
      ClientRequestInfo<TResult> info;

      if (_requests.TryRemove(id, out info))
      {
        switch (status)
        {
          case STaskStatus.Completed: info.CompletionSource.SetResult(
            (TResult)result);
            break;
          case STaskStatus.Canceled: info.CompletionSource.SetCanceled();
            break;
          case STaskStatus.Faulted: info.CompletionSource.SetException(
            (Exception)result);
            break;
        }
      }

}

  internal CallbackHandler(ConcurrentDictionary<string,
    ClientRequestInfo<TResult>> requests)
  {
    requests = requests;
  }
}
```

telerik

The 1st toolset enabling you to be successful in the marketplace or enterprise

Windows 8 UI Controls

Build for Windows 8 with either XAML or HTML

XAML HTML

www.telerik.com/win8

telerik
deliver more than expected

Figure 8 **AddRequest and SubmitRequest Methods**

```
public void AddRequest(string typeName, string[] parameters, CancellationToken tk)
{
  var info = new ClientRequestInfo<TResult>(typeName, args);

  _buffer.Add(info);

  tk.Register(()=> _client.CancelTask(info.TaskExecutionRequest.Id));

}

public void AddRequest(string typeName, string[] parameters)
  {
    _buffer.Add(new ClientRequestInfo<TResult>(typeName, parameters));
  }

public Task<TResult>[] SubmitRequests()
  {
    if (_buffer.Count == 0)
      return null;

    var req = _buffer.Select((r) =>
    {
      _requests.TryAdd(r.TaskExecutionRequest.Id, r);
      return r.TaskExecutionRequest;
    });

    _client.SubmitRequest(req.ToList<STask>());

    var ret =  _buffer.Select(r =>
      r.CompletionSource.Task).ToArray<Task<TResult>>();

    _buffer.Clear();

    return ret;
  }
```

Handling Client Requests

The task coordinator is the only point of entry for the clients, which means it must be able to handle as many client requests as possible while keeping the task execution nodes from becoming saturated (in terms of resources). This isn't as easy as it might seem. To better explain the potential challenges, let's look at a simplistic solution:

1. The task coordinator exposes the service operation through which the clients submit task execution requests.
2. The task coordinator submits these requests to the task execution nodes for their execution and keeps track of these requests—that is, it persists the state.

Figure 9 **The STaskInfo and CoordinatorContext Classes**

```
public class STaskInfo
  {
    public string ExecutionRequestId
    { get; set; }
    public STask ClientRequest
    { get; set; }
    public ITaskUpdateCallback CallbackChannel
    { get; private set; }
    public ITaskExecutionNode ExecutionRequestChannel
    { get; set; }

    public STaskInfo(ITaskUpdateCallback callback)
    {
      CallbackChannel = callback;
    }
  }

public static class CoordinatorContext
{
...
private static readonly ConcurrentDictionary<string, STaskInfo> _submissionTracker =
  new ConcurrentDictionary<string, STaskInfo>();

...
}
```

Figure 10 **Implementing the Submission Process**

```
public class TaskCoordinatorService : ITaskCoordinator
{
...

  public void SubmitRequest(List<STask> stasks)
  {
    CoordinatorContext.SendTasksToTaskHandler(stasks);
  }
...
}

public static class CoordinatorContext
{
...
  internal static void SendTaskRequestToTaskExecutionNode(List<STask> stasks)
  {
  var clientFactory = //Client factory creation logic..
  var channel = clientFactory.CreateChannel();

  foreach (var stask in stasks)
    _submissionTracker.TryAdd(stask.Id, stask);

  try
  {
    ((IClientChannel)channel).Open();

    channel.Start(stasks);

  }
  catch (CommunicationException ex)
  {
    // Error handling and logging ...
  }
  finally
  {
    if (((IClientChannel)channel).State != CommunicationState.Faulted)
       ((IClientChannel)channel).Close();
  }
}
...
}
```

Figure 10 shows a basic implementation of this submission process.

However, this simplistic implementation wouldn't work very well in some scenarios:

- If the client submits a large number of tasks in a single request, all of them will end up in a single task execution node, resulting in an uneven utilization of the available resources (assuming there's more than one task execution node available).
- In peak load scenarios, the system might exhaust the available resources in the task execution nodes if the number of executing TPL tasks exceeds what those resources can handle. This might be the case when what's been executed as a TPL task is bound to a particular resource (such as memory) that in peak cases can increase the risk of making the system unresponsive.

The Throttles

A way to address such challenges is to somehow "manage" the task execution requests as they go through the system. In this context, you can think of task coordinator as a throttling controller. Before I discuss the throttling process, however, let's review the semantics of the throttles that, in conjunction with the throttling process, I'll use to mitigate these risks.

The first scenario can be mitigated by capping the number of task execution requests the task coordinator can submit to the task

Total File Coverage

Aspose.Words
DOC, DOCX, RTF, HTML, PDF,
XPS & other document formats.

Aspose.Cells
XLS, XLSX, XLSM, XLTX, CSV,
SpreadsheetML & image formats.

Aspose.Pdf
PDF, XML, XLS-FO, HTML, BMP,
JPG, PNG & other image formats.

Aspose.Email
MSG, EML, PST, EMLX &
other formats.

and many more!

Aspose.BarCode Aspose.Tasks
Aspose.Slides Aspose.Diagram
Aspose.OCR Aspose.Imaging

Create
Modify
Convert
Combine
& Print

Scan our QR Code
for an exclusive
20% coupon code.

Follow us on
Facebook & Twitter

ASPOSE
Your File Format Experts

Get your FREE evaluation copy at **http://www.aspose.com**

US Sales: 1.888.277.6734
sales@aspose.com

EU Sales: +44 (0) 141 416 1112
sales.europe@aspose.com

AU Sales: +61 2 8003 5926
sales.asiapacific@aspose.com

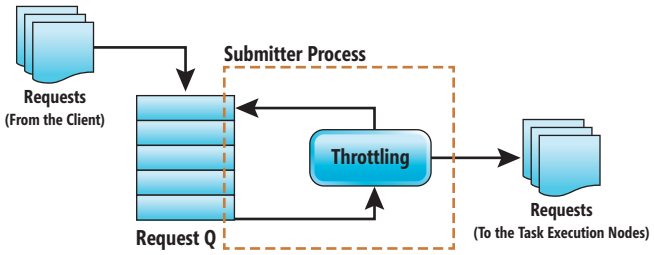


Figure 11 **The Submission Process**

execution nodes in a single request. I'll call this throttle maxSTasksPerRequest. Using this approach, the load balancer algorithm will be able to do its job of balancing the load across the available task execution nodes.

The second scenario is more challenging. A plausible solution is to cap the number of tasks the task execution nodes will execute at a particular number. I'll refer to this throttle as maxNumberOfTasks.

In addition to this throttle, the solution could benefit from having another throttle that limits the number of tasks being executed based on their type. To explain why this is useful, let's consider a scenario in which the task execution nodes have two types of tasks deployed, T1 and T2. T1 is CPU-bound and T2 is disk I/O-bound. In this scenario, the throughput of a client submitting requests for the execution of T1 tasks is more likely to be affected by active tasks that are bound by the same type of constraint—so the higher the number of T1 tasks the greater the impact. Because T2 tasks are bound by a different constraint, the impact they have on T1 tasks isn't the same. Having the ability to limit the execution of tasks by type means I can control how many T1 tasks can be running at any particular time, allowing me to maximize CPU resources and, as a result, the overall throughput. I will refer to this throttle as maxNumberOfTasksByType.

Queuing and Throttling

Now that you understand the semantics of the throttles and how throttles can be effective for maintaining the healthy operation of the task execution nodes, let's take a look at what happens when the limit specified by the throttles is reached—that is, the actual throttling process.

One option is to simply throw an exception. However, this would affect the overall throughput of the solution, because the client would incur the overhead of checking for a specific error or fault and then resubmitting the requests until the task coordinator could handle them successfully. An alternative would be to use server-side queuing to temporarily hold the requests from the client and a monitor-like process (a submitter process) that, at regular intervals, would read the requests from the queue and submit them to the task execution nodes. I'll use the submitter process to perform the actual throttling because the submitter reads from the request queue by considering the following rules:

1. Cap the number of requests that can be de-queued to maxSTasksPerRequest.
2. If the throttle maxNumberOfTasks is reached, stop de-queueing requests and the request queue will remain as is.
3. If the throttle maxNumberOfTasksByType is reached, de-queue and then enqueue the request back to the request queue. Enqueuing the request again allows the continuation of processing for tasks of other types. This strategy provides equal opportunity of execution for all the tasks in the queue. In some cases, however, you might consider using a priority queue instead. You'll find a good reference at bit.ly/NFOxQq.

Figure 11 illustrates this process.

Figure 12 **The SubmitRequest Service Operation**

```
[ServiceBehavior(ConcurrencyMode = ConcurrencyMode.Reentrant)]
public class TaskCoordinatorService : ITaskCoordinator
  {
    public void SubmitRequest(List<STask> stasks)
    {
      CoordinatorContext.EnqueueRequestsInRequestQ(stasks);
    }
    ...

}

public static class CoordinatorContext
{
...

internal static void EnqueueRequestsInRequestQ(List<STask> stasks)
  {
    var callback =
      OperationContext.Current.GetCallbackChannel<ITaskUpdateCallback>();

    foreach (var stask in stasks)
      _requestQ.Enqueue(new STaskInfo(callback) { ClientRequest = stask });
  }
...

}
```

I'll start describing the implementation of this process by showing the code (see **Figure 12**) for the SubmitRequest service operation that enqueues the requests in the request queue as it receives the requests from the client.

Next, let's look at the implementation of the submitter process, shown in **Figure 13**.

In **Figure 12** and **Figure 13**, you can see the service operation enqueing (writing) a request in the request queue and the submitter task de-queuing (reading) from the request queue. In this scenario, you need to ensure that the underlying data structure—the queue—is thread-safe. Fortunately, there's a class geared precisely for this,

Figure 13 **Submitter Implementation**

```
public static class CoordinatorContext
{
  ...
static CoordinatorContext()
  {
    Submitter(...);
  }

  private static async void Submitter(int interval)
  {
    while (true)
    {
      await Task.Delay(interval);
      SendTaskRequestToTaskExecutionNode(
      GetTasksFromRequestQ());
    }
  }
  ...
}
```

Figure 14 **GetTasksFromRequestQ Implementation**

```
public static class CoordinatorContext
{
  ...
internal static List<STaskInfo> GetTasksFromRequestQ()
{
  var ret = new List<STaskInfo>();
  var maxSTasksPerRequest = //From a configuration
  var maxNumberOfTasks = //From a configuration

  var count =  // Count of submitted or executing tasks
  var countByType = // Enumerable of count by type

  for (int i = 0; i < maxSTasksPerRequest; i++)
  {
    STaskInfo info;
    if (count + i == maxNumberOfTasks || !_requestQ.TryDequeue(out info))
      return ret;

    var countTT = // Count of submitted or executing tasks of
                  // the type of the current item

    if (countTT == GetMaxNumberOfTasksByType(info.ClientRequest.STaskTypeName))
    { _requestQ.Enqueue(info); }
    else ret.Add(info);
  }
  return ret;
}
}

private static int GetMaxNumberOfTasksByType(string taskTypeName)
{
  // Logic to read from a configuration repository the value by task type name
}
...
}
```

the ConcurrentQueue<T>. So I'll use a single instance of this type as the underlying repository for the requests.

```
public static class CoordinatorContext
{
  ...
private static readonly ConcurrentQueue<STaskInfo> _requestQ =
  new ConcurrentQueue<STaskInfo>();
  ...
}
```

Now, let's review the implementation of the method GetTasksFromRequestQ, which reads the tasks when the execution interval elapses. It's in this method that the throttling process occurs and where the throttles I described earlier apply. **Figure 14** shows an implementation of this process.

The goal of the implementation in **Figure 14** is obtaining the numbers that allow the process to assess the throttling conditions. **Figure 15** shows the plausible LINQ queries that can be executed against the _submissionTracker, as well as a list containing the

Figure 15 **The Throttling Values**

```
var countByType = (from t in _submissionTracker.Values
                   group t by t.ClientRequest.STaskTypeName into g
                   select new
                   {
                      TypeName =  g.Key,
                      Count = g.Count()
                   });

var count = countByType.Sum(c => c.Count);

var countTT = (from tt in countByType
               where tt.TypeName == info.ClientRequest.STaskTypeName
               select tt.Count).SingleOrDefault()+ ret.Where((rt) =>
rt.ClientRequest.STaskTypeName == info.ClientRequest.STaskTypeName)
                   .Count();
```

Figure 16 **The SendTaskToTaskExecutionNode and Supporting Methods**

```
internal static void SendTaskRequestToTaskExecutionNode(List<STaskInfo> staskInfos)
  {

  if (staskInfos.Count() == 0)
    return;

  var channel = new DuplexChannelFactory<ITaskExecutionNode>(
                new InstanceContext(new CallbackHandler()),
                new NetHttpBinding(), new EndpointAddress("http://.../"))
                .CreateChannel();
  try
  {
    var requestId = Guid.NewGuid().ToString();
    var reqs = staskInfos.Select(s => AddRequestToTracker(requestId,s, channel))
      .Where(s => s != null);

    ((IChannel)channel).Open();

    channel.Start(reqs.ToList<STask>());
  }
  catch (CommunicationException ex)
  {
    foreach (var stask in staskInfos)
      HandleClientUpdate(stask.ClientRequest.Id, STaskStatus.Faulted, ex);
  }

  }

private static STask AddRequestToTracker(string requestId,
  STaskInfo info, ITaskExecutionNode channel)
{
  info.ExecutionRequestId = requestId;
  info.ExecutionRequestChannel = channel;

  if (_submissionTracker.TryAdd(info.ClientRequest.Id, info))
    return info.ClientRequest;

  HandleClientUpdate(info.ClientRequest.Id, STaskStatus.Faulted,
    new Exception("Failed to add "));

  return null;
}
```

return items (ret) to obtain these values. Note that this approach might succeed at the expense of performance. If so, as an alternative you could implement a set of thread-safe counters that increment or decrement as items are added or removed from the submission tracker instance and use those counters instead of querying the concurrent dictionary directly.

Sending Requests to the Task Execution Nodes and Handling Results

So far I've discussed how the task coordinator manages the requests. Let's look at how the task coordinator submits the request to the task execution nodes, now considering the throttling process. To provide better context, let's first review the service operations that the task execution nodes expose (through the load balancer):

```
[ServiceContract( CallbackContract = typeof(ITaskUpdateCallback))]
  public interface ITaskExecutionNode
  {
    [OperationContract]
    void Start(List<STask> stask);

    [OperationContract]
    void Cancel(string Id);
  }
```

As their names suggest, the purposes of these operations are to start a list of task execution requests and to request cancellation of a particular task. The service contract leverages the same

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

Figure 17 **HandleClientUpdate and Supporting Methods**

```
internal async static void HandleClientUpdate(
  string staskId, STaskStatus status, object result)
  {
    STaskInfo info;

    if (!_submissionTracker.TryGetValue(staskId, out info))
      throw new Exception("Could not get task from the tracker");

try
  {

    await Task.Run(() =>
      info.CallbackChannel.UpdateStatus(info.ClientRequest.Id, status, result));

    RemoveComplete(info.ClientRequest.Id);
  }
catch(AggregateException ex)
  {
  // ...
  }
  }

private static void RemoveComplete(string staskId)
  {
    STaskInfo info;
    if (!_submissionTracker.TryRemove(staskId, out info))
      throw new Exception("Failed to be removed from the tracking collection");

    if (_submissionTracker.Values.Where((t) => t.ExecutionRequestId ==
      info.ExecutionRequestId).Count() == 0)
      CloseTaskRequestChannel((IChannel)info.ExecutionRequestChannel);
  }

  private static void CloseTaskRequestChannel(IChannel channel)
  {
    if (channel != null && channel.State != CommunicationState.Faulted)
      channel.Close();
  }
```

callback contract to update the task coordinator via an implementation of the contract.

Figure 16 shows an updated implementation of the method SendTaskToTaskExecutionNode where the task coordinator stores the STaskInfo instances in the _submissionTracker and calls the Start service operations on a task execution node.

Note that the method SendTaskToTaskExecutionNode creates a callback instance to handle the result of the execution of the task in a task execution node:

```
[ServiceBehavior(ConcurrencyMode = ConcurrencyMode.Reentrant)]
  public class CallbackHandler : ITaskUpdateCallback
  {
    public void UpdateStatus(string id, STaskStatus status, string result)
    {
      CoordinatorContext.HandleClientUpdate (id, status, result);
    }
  }
```

The CallbackHandler handles the callback operation by calling the HandleClientUpdate method. This method retrieves and removes the corresponding STaskInfo instance from the submitterTracker and performs a callback to the client to update the result. In addition, if this is the last request in the group, it closes the channel between the task coordinator and the task execution node. **Figure 17** shows the implementation of the HandleClientUpdate method.

Task Implementer

In the client code, typeName is one of the required parameters when adding requests. This value eventually makes it to the task execution node. The value of typeName is the type name of the implementation of an interface that exposes a function delegate that encapsulates the functionality intended to run as a parallel task and that resides in all the task execution nodes. I'll call this interface IRunnableTask. Implementers of this interface should expect to receive as parameters a cancellation token and an array of parameters from the client. The delegate should also return the result of the task. Here's the interface:

```
public interface IRunnableTask
{
  Func<Object> Run(CancellationToken ct, params string[] taskArgs );
}
```

Starting a Task in a Task Execution Node

At a high level, a task execution node is responsible for "transforming" a task execution request into an actual task that the TPL can execute—that is, starting a TPL task. **Figure 18** shows an implementation of this process, which I'll discuss.

Step 1 (a and b): At this stage, the task execution node needs to create an instance of IRunnableTask, which will return a delegate that will run as a task of the type requested by the client. For this, I leverage MEF and a new feature in the .NET Framework

Figure 18 **Starting a Task**

```
[ServiceBehavior(ConcurrencyMode = ConcurrencyMode.Reentrant)]
public class TaskExecutionNodeHandler : ITaskExecutionNode
{
  public void Start(List<STask> stasks)
  {

    var callback =
      OperationContext.Current.GetCallbackChannel<ITaskUpdateCallback>();
    foreach (var t in stasks)
      TaskExecutionContext.Start(t,callback);

  }
...
}

public static class TaskExecutionContext
{
...
internal static void Start(STask stask, ITaskUpdateCallback callback)
{

  try
  {
    // Step 1.a
    var rtasks = CompositionUtil.ContainerInstance.GetExports<IRunnableTask>();

    // Step 1.b
    var rtask = from t in rtasks
                where t.Value.GetType().FullName == stask.STaskTypeName
                select t.Value;

    // Step 2
    var cs = new CancellationTokenSource();
    var ct = cs.Token;
    TaskExecutionContext._cancellationSources.TryAdd(stask.Id, cs);

    // Step 3
    Task<Object>
      .Run(rtask.First().Run(ct, stask.STaskParameters), ct)
      .ContinueWith(tes => UpdateStatus(tes, stask, callback));

  }
  catch (Exception ex)
  {
    ...
  }
}
...
}
```

4.5 that allows an attribute-free configuration approach. The code in **Figure 19** creates a single container instance that exports all the implementations of IRunnableTask located in the directory "extensions." For more information about MEF and the attribute-free configuration approach, please see the June 2012 *MSDN Magazine* article, "An Attribute-Free Approach to Configuring MEF" at msdn.microsoft.com/magazine/jj133818.

Now let's go back to the code in **Figure 18**. The code uses the container to get the exports of the type IRunnableTask, then selects the instance with the type name that matches the client request. Note that I make the key assumption that there's only one task instance that corresponds to the type requested by the client. This is the reason I use the first instance that the LINQ query returns.

Figure 19 **Creating a Container**

```
internal static  class CompositionUtil
{
  private readonly static Lazy<CompositionContainer> _container =
    new Lazy<CompositionContainer>(() =>
    {
      var builder = new RegistrationBuilder();
      builder.ForTypesDerivedFrom<IRunnableTask>()
             .Export<IRunnableTask>()
             .SetCreationPolicy(CreationPolicy.NonShared);


      var cat = new DirectoryCatalog("extensions", builder);

      return new CompositionContainer(cat, true, null);
    }
  ,true);

  internal static CompositionContainer ContainerInstance
  {
    get { return _container.Value; }
  }
}
```

Figure 20 **Encapsulating the Update Process**

```
  private static Action<Task<Object>, STask, ITaskUpdateCallback>
  UpdateStatus = (t, st, cb) =>
    {
      try
      {
        STaskStatus s;
        Object r = null;
        switch (t.Status)
        {
          case TaskStatus.Canceled: s = STaskStatus.Canceled;
            break;
          case TaskStatus.Faulted:
            s = STaskStatus.Faulted;
            r = t.Exception.Flatten();
            break;
          case TaskStatus.RanToCompletion:
            s = STaskStatus.Completed;
            r = t.Result;
            break;
          default:
            s = STaskStatus.Faulted;
            r = new Exception("Invalid Status");
            break;
        }
        CancellationTokenSource cs;
        TaskExecutionContext._cancellationSources.TryRemove(st.Id, out cs);

        cb.UpdateStatus(st.Id, s, r);
      }
      catch (Exception ex)
    {
    // Error handling
    }
  };
```

Step 2: Before actually creating the TPL task, the code creates a Cancellation Token Source and a Cancellation Token. I'll keep track of the Cancellation Source in a single instance of a ConcurrentDictionary<TKey,TValue>. The task execution node will use this list of cancellation sources when a client requests a cancellation. Here's the definition of this instance:

```
public static class TaskExecutionContext
{
...
private readonly static ConcurrentDictionary<string,
  CancellationTokenSource> _cancellationSources =
  new ConcurrentDictionary<string, CancellationTokenSource>();

...
}
```

Step 3: At this point, I run the task, with the cancellation token I just created. The task is followed by a continuation task.The need for a continuation task arises because the task coordinator must be updated, by making a service call, with the result of the execution once the TPL task has completed (successfully or with a fault). As **Figure 20** shows, I encapsulate the process of updating the task coordinator in a delegate that receives as a parameter the TPL task, the task execution request and a callback instance to the task coordinator.

Requesting and Handling a Cancellation

The TPL provides a mechanism for implementing task cancellation. For this, the delegate that encapsulates the actual process that's running as a TPL task needs to respond to the cancellation request and terminate the execution. For more information about task cancellation, see the MSDN Library article, "Task Cancellation," at bit.ly/NYVT0O.

One of the parameters in the IRunnableTask interface is a cancellation token. The task execution node will create a token for each task and it's up to the implementer of the interface to determine when to check for a cancellation request and terminate the process gracefully. The code in **Figure 21** shows a simple task that calculates the number of even numbers in a range, while checking if a cancellation has been requested.

As you saw when I discussed the client, you can add a request with a cancellation token and internally the client performs the necessary subscription. So when a cancellation is raised, a

Figure 21 **Checking for a Cancellation**

```
public class MySimpleCTask : IRunnableTask
{
  public Func<Object> Run(Nullable<CancellationToken> ct, params string[] taskArgs)
  {
    var j = int.Parse(taskArgs[0]);
    var z = 0;
    return (() =>
  {
      for (int i = 0; i < j; i++)
      {
        if (i % 2 != 0)
        {
          z++;

          ct.Value.ThrowIfCancellationRequested();
        }
      }

      return z;
   });
  }
}
```

Figure 22 **Implementing the Service Operations in the Task Coordinator**

```
public class TaskCoordinatorService : ITaskCoordinator
{
...
public bool CancelTask(string Id)
  {
    return CoordinatorContext.CancelTask(Id);
  }
  ...
}
public static class CoordinatorContext
{
...
internal static bool CancelTask(string Id)
{
STaskInfo info;
if(_submissionTracker.TryGetValue(
  Id, out info) && info.ExecutionRequestChannel != null)
{
  info.ExecutionRequestChannel.Cancel(Id);
  return true;
}
return false;

}
  ...
}
```

cancel request is sent to the task coordinator. Upon receiving the cancellation request, the task coordinator checks whether the request has been submitted to a task execution node and sends a cancellation request. The task execution node then looks for the cancellation source that corresponds to the task requested by the client Id. Submitting the cancellation request to the task execution node is relatively simple—you just need to locate the channel that corresponds to the request where the task coordinator initially submitted the task execution request. These channels need to be kept open for the callbacks that update the status of the execution request.

Figure 22 shows the implementation of the service operations in the task coordinator.

Finally, **Figure 23** shows the implementation of the service operations in the task execution nodes.

Scalability of the Task Coordinator and Other Considerations

It's worth noting that this implementation assumes the task coordinator runs on a single node, but it's quite possible to scale out the task coordinator (this would require, at the least, the following changes):

- A load balancer for accessing the task coordinator would need to be introduced.
- As I described earlier, the key to the throttling approach is having an accurate count of the number of tasks that are running, in total and by type. In a scenario with more than one node running as task coordinators, these counters will need to be maintained centrally (for example, in a database) while still being able to be updated or read in a synchronized way (avoiding race conditions, deadlocks and so forth).

Finally, let me note that as with any development approach, the risk and value need to be weighed against other alternatives that might meet your needs and that are available off the shelf. For instance, you might want to consider technologies such as the

Figure 23 **Implementing the Service Operations in the Task Execution Nodes**

```
class CancellationHandler : ICancellationHandler
  {
  public void Cancel(STask stask)
  {
    TaskExecutionContext.CanceTask(stask.Id);
  }
  }

public static class TaskExecutionContext
{
...

  internal static void CancelTask(string Id)
  {
    CancellationTokenSource tknSrc;
    if (_cancellationSources.TryGetValue(Id, out tknSrc))
      tknSrc.Cancel(); }
...

}
```

Microsoft HPC server as a plausible solution for many scenarios that you otherwise might think of addressing based on the approach described in this article.

Optimizing Resources

The TPL provides the necessary infrastructure to achieve the most optimal utilization of CPU resources within a single multicore computer, and it's also useful for implementing an approach that scales across computer boundaries. This can be helpful for workload automation and batch-processing scenarios where parallelism is required not only in a single multicore server, but across multiple servers as well.

The key to the throttling approach is having an accurate count of the number of tasks that are running, in total and by type.

To achieve this horizontal scalability, several architectural considerations need to be taken into account. Key among them: the need to balance the load across the existing resources while having the ability to add more resources to the existing farm, and the ability to throttle the resources according to the semantics of the tasks that need to be executed. Microsoft development tools and technologies provide the necessary building blocks to implement an architecture that takes into account these key considerations. ■

Jesus Aguilar *works for Microsoft in the area of Premier Support for Developers as a senior application development manager.*

Thanks to the following technical experts for reviewing this article: *Ryan Berry, Steve Case, Rick Claude and Piyush Joshi*

Compatible with
Microsoft® Visual Studio® 2012

FREE TRIAL DOWNLOAD
INFRAGISTICS.COM/DOWNLOADS

GET A PULSE

On Mobile Business Intelligence

infragistics.com/ **EXPERIENCE**

INFRAGISTICS
DESIGN / DEVELOP / EXPERIENCE

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

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

YOUR MAP TO THE .NET DEVELOPMENT PLATFORM

BIRDS OF A FEATHER CODE TOGETHER

Orlando, FL | **December 10-14**

Royal Pacific Resort at Universal Orlando | **vslive.com/orlando**

Don't miss your chance for great .NET education in 2012 at Visual Studio Live! Orlando.

- **Mobile**
- **Visual Studio / .NET**
- **Web / HTML5**
- **Windows 8 / WinRT**
- **WPF / Silverlight**

Save Up To $400! Register Before November 7

Use Promo Code VSL2OCT

vslive.com/orlando

ROYAL PACIFIC RESORT LIVE! DECEMBER 10 - 14 LIVE! ORLANDO

Visit **vslive.com/orlando** or scan the QR code to register and for more event details.

GOLD SPONSOR: axosoft

SUPPORTED BY: Microsoft, Microsoft Visual Studio, msdn, Visual Studio MAGAZINE

PRODUCED BY: 1105 MEDIA

MEDIA SPONSOR: THE CODE PROJECT WWW.CODEPROJECT.COM

VISUAL STUDIO LIVE! ORLANDO AGENDA AT-A-GLANCE

Web / HTML 5	Mobile	WPF / Silverlight	Windows 8 / WinRT	Visual Studio / .NET

Start Time	End Time	Visual Studio Live! Pre-Conference Workshops: Monday, December 10, 2012 *(Separate entry fee required)*			
8:00 AM	12:00 PM	VSM1 - Workshop: XAML User Experience Design *Billy Hollis*	VSM2 - Workshop: Services - Using WCF and ASP.NET Web API *Miguel Castro*		VSM3 - Workshop: Build a Windows 8 Application in a Day *Rockford Lhotka*
1:00 PM	5:00 PM	VSM1 - Workshop: XAML User Experience Design *Billy Hollis*	VSM2 - Workshop: Services - Using WCF and ASP.NET Web API *Miguel Castro*		VSM3 - Workshop: Build a Windows 8 Application in a Day *Rockford Lhotka*
5:00 PM	7:00 PM	EXPO Preview			
7:00 PM	8:00 PM	Live! 360 Keynote			

Start Time	End Time	Visual Studio Live! Day 1: Tuesday, December 11, 2012			
8:00 AM	9:00 AM	Visual Studio Live! Keynote			
9:15 AM	10:30 AM	Live! 360 Keynote			
11:00 AM	12:15 PM	VST1 **Controlling ASP.NET MVC 4** *Philip Japikse*	VST2 **Leveraging XAML for a Great User Experience** *Billy Hollis*	VST3 **Intro to Metro** *Miguel Castro*	VST4 **What's New in Visual Studio 2012** *Rob Daigneau*
2:00 PM	3:15 PM	VST5 **Azure Web Hosting (Antares)** *Vishwas Lele*	VST6 **Implementing the MVVM Pattern in WPF** *Miguel Castro*	VST7 **Smackdown: Metro Style Apps vs. Websites** *Ben Dewey*	VST8 **What's New in Windows Phone 8 for Developers** *Nick Landry*
4:15 PM	5:30 PM	VST9 **I'm Not Dead Yet! AKA the Resurgence of WebForms** *Philip Japikse*	VST10 **Building High Performance, Human-Centered Interactive Data Visualizations Dashboards** *Nick Landry*	VST11 **Going from Silverlight or WPF to Metro** *Greg Levenhagen*	VST12 **What's New in the .NET 4.5 BCL** *Jason Bock*
5:30 PM	7:30 PM	Exhibitor Reception			

Start Time	End Time	Visual Studio Live! Day 2: Wednesday, December 12, 2012			
8:00 AM	9:00 AM	Visual Studio Live! Keynote			
9:15 AM	10:30 AM	VSW1 **JavaScript and jQuery for .NET Developers** *John Papa*	VSW2 **Navigation and UI Shells for XAML Applications** *Billy Hollis*	VSW3 **Filling Up Your Charm Bracelet** *Ben Dewey*	VSW4 **Coming Soon!**
11:00 AM	12:15 PM	VSW5 **Applying JavaScript Patterns** *John Papa*	VSW6 **Static Analysis in .NET** *Jason Bock*	VSW7 **WinRT Services** *Rockford Lhotka*	VSW8 **Reach the Mobile Masses with ASP.NET MVC 4 and jQuery Mobile** *Keith Burnell*
1:45 PM	3:00 PM	VSW9 **HTML5 for Business: Forms and Input Validation** *Todd Anglin*	VSW10 **What's New in Azure for Devs** *Vishwas Lele*	VSW11 **Using Azure with Windows Phone and Windows 8!** *Greg Levenhagen*	VSW12 **iOS Development Survival Guide for the .NET Guy** *Nick Landry*
4:00 PM	5:15 PM	VSW13 **Making HTML5 Work Everywhere** *Todd Anglin*	VSW14 **In Depth Azure PaaS** *Vishwas Lele*	VSW15 **From a New Win8 Project to the Store** *Elad Shaham*	VSW16 **Creating Restful Web Services with the Web API** *Rob Daigneau*

Start Time	End Time	Visual Studio Live! Day 3: Thursday, December 13, 2012			
8:00 AM	9:15 AM	VSTH1 **Identify & Fix Performance Problems with Visual Studio Ultimate** *Benjamin Day*	VSTH2 **Patterns for Parallel Programming** *Tiberiu Covaci*	VSTH3 **Windows 8 Metro Style Apps for the Enterprise** *Ben Hoelting*	VSTH4 **How to Take WCF Data Services to the Next Level** *Rob Daigneau*
9:30 AM	10:45 AM	VSTH5 **Building Single Page Web Applications with HTML5, ASP.NET MVC4, Upshot.js and Web API** *Marcel de Vries*	VSTH6 **ALM Features in VS12** *Brian Randell*	VSTH7 **Build Windows 8 Apps using Windows Online Services** *Rockford Lhotka*	VSTH8 **Coming Soon!**
11:00 AM	12:15 PM	VSTH9 **Busy .NET Developer's Guide to Node.js** *Ted Neward*	VSTH10 **Intellitrace, What is it and How Can I Use it to My Benefit?** *Marcel de Vries*	VSTH11 **Expression Blend 5 For Developers : Design Your XAML or HTML5\CSS3 UI Faster** *Ben Hoelting*	VSTH12 **Coming Soon!**
1:30 PM	2:45 PM	VSTH13 **Cloudant NoSQL on Azure** *Sam Bisbee*	VSTH14 **Team Foundation Server 2012 Builds: Understand, Configure, and Customize** *Benjamin Day*	VSTH15 **Making your Windows 8 App Come to Life, Even When It's Not Running** *Elad Shaham*	VSTH16 **EF Code First Magic Unicorn Edition and Beyond** *Keith Burnell*
3:00 PM	4:15 PM	VSTH17 **Introduction to OAuth** *Ted Neward*	VSTH18 **Azure Hosted TFS** *Brian Randell*	VSTH19 **No User Controls Please: Customizing Metro Style Controls using XAML** *Elad Shaham*	VSTH20 **The LINQ Programming Model** *Marcel de Vries*
4:30 PM	5:45 PM	Live! 360 Conference Wrap-up			

Start Time	End Time	Visual Studio Live! Post-Conference Workshops: Friday, December 14, 2012 *(Separate entry fee required)*	
8:00 AM	12:00 PM	VSF1 **Workshop: Mastering ASP.NET MVC4 in Just One Day** *Tiberiu Covaci*	VSF2 **Workshop: TFS/ALM** *Brian Randell*
1:00 PM	5:00 PM	VSF1 **Workshop: Mastering ASP.NET MVC4 in Just One Day** *Tiberiu Covaci*	VSF2 **Workshop: TFS/ALM** *Brian Randell*

For the complete session schedule and full session descriptions, please check the Visual Studio Live! Orlando web site at ***vslive.com/orlando***
**Speakers and Sessions Subject to Change.*

OData, the Entity Framework and Windows Azure Access Control

Sean Iannuzzi

In this article I'll show how to implement the Open Data Protocol (OData) with the Entity Framework exposed with Windows Communication Foundation (WCF) RESTful services and secured with the Windows Azure Access Control Service (ACS).

Like most developers, I often find myself trying to leverage a combination of technologies in new and various ways to complete my project as efficiently as possible while also providing a flexible, easy-to-maintain solution. This can be difficult, particularly when the project requires data to be exposed quickly and securely.

Recently I was asked to create a secure Web service for an existing database and Web application. I really didn't want to implement all of the create, read, update and delete (CRUD) code. It was tempting to just create custom service contracts, operation contracts and data contracts that would drive exactly how the data could be exposed and how someone else could potentially consume this data via services. But I knew there had to be a more advantageous route to take. I started researching various ways that this could be accomplished and saw potential with OData (or "Ohhhh Data," as I like to call it). The problem was that OData by itself was not secured in a way that I felt was acceptable, and I needed to add an additional layer of security on top of the OData service in order to feel confident it would be secured properly. As I started to piece this together, I found ACS, which is great for implementing a cloud-based federated authentication and authorization service—exactly what I needed. Then I had my "aha!" moment. I realized that if I wired up ACS with OData, I'd have my solution.

Now, I did consider implementing custom service contracts, and there is a place for this approach, especially where a layer of abstraction is needed in front of a data model and where it's required to protect the database entities from being directly exposed to consumers of a service. However, given how time-consuming this is—creating the appropriate document regarding how to consume the service, and adding in the additional effort required to set up the security ("MessageCredential" and "TransportWithMessageCredentials")—the project could quickly spiral of control. I was also concerned that additional methods would be needed or requested for one reason or another to support how the services were being

This article discusses:

- Setting and configuring Windows Azure Access Control Service
- Creating a Secure OData WCF Data Service with the Entity Framework
- Creating a custom Windows Azure Access Control Service module
- Consuming the OData WCF RESTful service in .NET and Java

Technologies discussed:

OData, Windows Azure Access Control Service, Entity Framework, Windows Communication Foundation RESTful services

Code download available at:

archive.msdn.microsoft.com/mag201210OData

consumed, which, again, would add more time, maintenance and customization. Even if my implementation of the service used the Entity Framework versus ADO.NET directly, creating all of the CRUD code might still be required to keep the data layer in sync. Assuming there are a few dozen tables, this effort could be extremely tedious. Plus, creating and maintaining any additional documentation and implementation details required for end users to consume my service made this a much more complicated proposition to manage.

An Easier Way

Once I had identified the primary technologies, I looked for others to fill gaps and help build a cohesive solution. The goal was to limit the amount of code that needed to be written or maintained while securely exposing my OData WCF RESTful services. The technologies I linked together are: ACS, OData, Entity Data Models, WCF Data Services with Entity permissions and a custom Windows Azure security implementation. Each of these technologies already provides significant value on its own, but when combined, their value increases exponentially. **Figure 1** demonstrates a high-level overview of how some of these technologies will work when implemented.

Before I tried to combine all of these technologies, I had to take a step back and really understand each one and how they could impact this project. I then gained a good perspective on how to combine them all and what would be required from someone else using other technologies to consume my services.

What Is ACS?

ACS is provided as a component of the Windows Azure platform. ACS allows me to set up my own cloud-based federated authentication and authorization provider that I use to secure my OData WCF services, but ACS can also be used for securing any app. ACS is a cloud-based service that helps bridge the security gap when there's a need to implement single sign-on (SSO) in multiple applications, services or products—either cross-platform or cross-domain—supporting various SSO implementations. A Windows Azure account provides access to much more information. You can sign up for a free trial at windowsazure.com. To read more about ACS, see bit.ly/zLplbw.

What Is OData and Why Would I Use It?

OData is a Web-based protocol for querying and updating data, and exposing the data using a standardized syntax. OData leverages technologies such as HTTP, XML, JSON and the Atom Publishing Protocol to provide access to data differently. Implementing OData with the Entity Framework and WCF Data Services provides many great benefits.

I started to wonder why I would use this versus custom WCF contracts. The answer was straightforward. The most practical reason was to leverage service documentation that's already available and use a standardized syntax supporting how to access data from my service. Having written dozens of services, it seems that I always need to add an additional method as a result of providing custom services. And consumers of custom services tend to ask for more and more features.

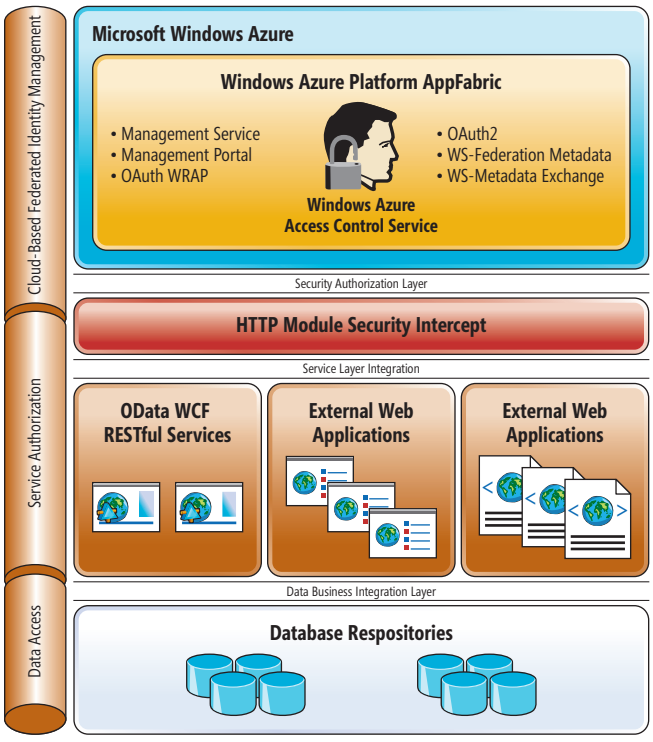


Figure 1 **High-Level Overview of ACS with a Security Intercept**

For more information on OData and OData URI conventions, visit the following sites:

- OData main site: odata.org
- OData URI conventions: bit.ly/NRalFb
- Netflix OData example service: bit.ly/9UZjRd

OData with the Entity Framework and WCF Data Services

Using OData with WCF Data Services and the Entity Framework exposes standard functionality to support the retrieval and saving of data in a way that's already documented with little implementation code. When I first started creating my Entity Data Model for Data Services Packaging Format (EDMX) and linked it to my WCF service through data services, I was very skeptical. However, it worked

Figure 2 **Implementing OData WCF Data Services**

```
using System.Data.Services;
using System.Data.Services.Common;
namespace WCFDataServiceExample
{
  public class NorthwindService : DataService<NorthwindEntities>
  {
    // This method is called only once to initialize service-wide policies.
    public static void InitializeService(DataServiceConfiguration config)
    {
      // Give full access to all of the entities.
      config.SetEntitySetAccessRule("*", EntitySetRights.All);
      // Page size will change the max number of rows returned.
      config.SetEntitySetPageSize("*", 25);
      config.DataServiceBehavior.MaxProtocolVersion =
        DataServiceProtocolVersion.V2;
    }
  }
}
```

Figure 3 **Technologies Used and Why**

Technology	Why Use It
ACS	Provides a way to secure services using a cloud-based federated security module for authentication and authorization
OData	Provides a standard syntax for querying and updating data while leveraging common technologies such as HTTP, JSON, the Atom Publishing Protocol and XML
Entity Data Models	Provide a quick way to create a common data access for a database tier while also providing serializable data contracts of tables in a database
WCF Data Services with Entity Permissions	Exposes the Entity Framework data contract with the appropriate level of permissions for CRUD as a WCF RESTful service
Custom Windows Azure Security Implementation	Secures services (in this case OData services) from being consumed without the proper level of security being applied, such as a token or certificate

perfectly. All of the entities I included in my EDMX were automatically included and exposed in my WCF service in a RESTful implementation. **Figure 2** shows some example code.

I created an EDMX and linked that to a few of the tables in my database (Northwind Sample Database). I then linked my database entities to my WCF Data Service, exposing all of the entities by using the "SetEntitySetAccessRule" method with a wildcard attribute of "*" for all entities. This allowed me to set various permissions on my entities for read, write and query access as well as set the page size, as shown in **Figure 2**. The code shown is the entire implementation of the OData WCF Data Services code.

The service contracts, operation contacts and data contracts are mainly controlled via the configuration in the initialization of the service being provided. In the initialization method I have the ability to set various permissions related to how I'd like to expose my entities and what level of access I'd like to provide to anyone looking to consume my services. I could even leverage T4 templates to create an abstract layer or template layer on top of the entities with custom entity names. This would provide an added level of clarity to the consumer of my services. I could even set the permission for a specific table, or I could set the table name along with the appropriate security settings for even lower-level protection. Here's an example of giving read access to the Customer table:

```
config.SetEntitySetAccessRule("Customer",
  EntitySetRights.AllRead);
```

Many different security implementations can be enabled with OData and WCF Data Services, but for now I'm only concerned with how to protect my WCF services using ACS in combination with the data service's access rules.

A quick list of the technologies used is shown in **Figure 3**, along with some of the reasons why these technologies would be used.

With each of these technologies there are always trade-offs based on the project, but I found that combining them saved up-front setup time and reduced maintenance efforts, while requiring less code and providing huge gains—especially when I needed to expose my data securely and supply common data-access methods with a standardized syntax.

Putting It All Together

With a fairly good understanding of the combined use of OData, the Entity Framework and WCF Data Services, I could apply some additional security features to this technology by leveraging ACS. There were a few options to secure my service from being accessed, including setting various permissions on the entities or adding query interceptors to prevent service consumption or control how my service could be consumed.

However, implementing query interceptors or setting permissions would be tedious, and adding a layer of security on top of my service to protect it from being consumed was preferred rather than writing additional code. Implementing a common security

Figure 4 **ACS Setup**

Configuration	Values
ACS Namespace	WCFoDataACS
Replying Party Application	Name: wcfodataacsexampleDevelopment Mode: Leave defaults (enter settings manually) Realm: http://localhost/WCFDataServiceExample/<servicename>.svc Return URL: Leave default (blank) Error URL: Leave default (blank) Token Format: SWT Token Encryption Policy: Leave default (none) Token Lifetime: Leave default (600)
Authentication Settings	Identity Providers: Uncheck Windows Live ID Rules Group: Check Create New Rule Group *Note: I created different settings for development, test and production.*
Token Signing Settings	Click on the Generate button Effective Date: Leave defaults Expiration Date: Leave defaults
Rule Group	*Note: Based on my settings, the rule group will be created automatically, but I'll still need to add the claim configuration.*
Claim Configuration	If Section: Access Control System: Selected Input claim type: Leave default (any) Input claim value: Leave default (any) Then Section: Output claim type: Leave default (pass through input claim type) Output claim value: Leave default (pass through input claim value) Rule Information: Description: Leave default or enter a description
Service Identity Within Windows	Name: the username to provide others (in this example I used wcfodataacsDevUser) Description: Leave defaults (or enter a description for the user) Realm: http://localhost/WCFDataServiceExample/<servicename>.svc Credential Settings: Type: Select password Password: Enter the desired password Effective Date: Leave defaults Expiration Date: Leave defaults *Note: There are several options for how to authenticate a user into the services created, but for simplicity, I used a password entry as the credential type. There are other options available—such as using a x509 certificate or a symmetric key—that may provide a higher level of security, but for this example I tried to keep it basic.*

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

Figure 5 **Security Validation**

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Web;
using Microsoft.AccessControl2.SDK;
namespace Azure.oAuth.SecurityModule
{
  internal class SWTModule : IHttpModule
  {
    // Service namespace setup in Windows Azure
    string _serviceNamespace = ConfigurationManager.AppSettings["serviceNamespace"];

    // ACS name
    string _acsHostName = ConfigurationManager.AppSettings["acsHostname"];

    // The key for which the service was signed
    string _trustedSigningKey =
      ConfigurationManager.AppSettings["trustedSigningKey"];

    // The URL that was setup in the rely party in Windows Azure
    string _trustedAudience = ConfigurationManager.AppSettings["trustedAudience"];

    // Setting to allow the metadata to be shown
    bool _enableMetaData =
       Convert.ToBoolean(ConfigurationManager.AppSettings["enableMetadata"]);

    // Setting to disable or enable the security module
    bool _disableSecurity =
      Convert.ToBoolean(ConfigurationManager.AppSettings["disableSecurity"]);

    const string _metaData = "$metadata";

    private void context_BeginRequest(object sender, EventArgs e)
    {
      if (!_disableSecurity)
      {
        string tempAcceptableURLs = String.Empty;
        // Check if the audiencename has trailing slash
        tempAcceptableURLs = _trustedAudience.ToLower();
        if (tempAcceptableURLs.Substring(_trustedAudience.Length - 1, 1) == "/")
        {
          tempAcceptableURLs =
            tempAcceptableURLs.Substring(0, _trustedAudience.Length - 1);
        }
        // First check if the person is requesting the WSDL or .svc
        if (_enableMetaData != false
          && HttpContext.Current.Request.Url.AbsoluteUri.ToLower() !=
          tempAcceptableURLs
          && HttpContext.Current.Request.Url.AbsoluteUri.ToLower() !=
          tempAcceptableURLs + _metaData
          && HttpContext.Current.Request.Url.AbsoluteUri.ToLower() !=
          tempAcceptableURLs + "/"
          && HttpContext.Current.Request.Url.AbsoluteUri.ToLower() !=
          tempAcceptableURLs + "/" + _metaData)
        {
          // SWT Validation...
          // Get the authorization header
          string headerValue =
            ttpContext.Current.Request.Headers.Get("Authorization");

          // Check that a value is there
          if (string.IsNullOrEmpty(headerValue))
          {
            throw new ApplicationException("unauthorized-1.1");
          }

          // Check that it starts with 'WRAP'
          if (!headerValue.StartsWith("WRAP "))
          {
            throw new ApplicationException("unauthorized-1.2");
          }
          // ... <code truncated> ...
        }
      }
    }
  }
}
```

mechanism that allows trusted parties or external companies to access my services would be ideal. In turn, I could use a combination of this security along with the entity protection to give my services the most secure implementation with the most flexibility.

Using this approach would require any consumer of my service to first authenticate through ACS and obtain a valid access token. The services would be restricted without this access token. Valid access tokens would be required in the request header before anyone would be granted access to my service. Once the consumer of my service was authorized, I'd apply fine-grained security on the entities to ensure only authorized access to the data or my entities.

ACS Setup and Configuration

Some setup and configuration is required in order to implement ACS. **Figure 4** shows a list of items that I set up for this example.

After completing the setup of ACS, I was able to secure my OData WCF RESTful services. Before I could secure them, I first had to implement a custom security module that could intercept the requests and validate the security to prevent unauthorized access.

ACS Security Implementation

As an example, I implemented the security using a custom HTTP module. This intercepts any request made to my service and validates if the proper authentication and authorization have occurred. Without this HTTP module, my services would be secure only at the entity level, based on the settings in the data service configuration.

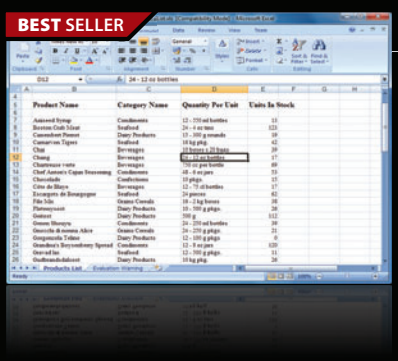
In this case, I secured these services with ACS; therefore, requests were intercepted and then checked for the proper level of security to make sure the consumer of the service had been granted the proper level of authorization. As noted earlier, fine-grained security could be implemented at the entity level after the consumer of the service had been authorized access.

When implementing the IHTTPModule interface, I chose to add some additional features, so I exposed parts of the service metadata in order to allow consumers of the service to auto-generate classes (similar to the behavior of adding any other Web service). I added these sections of code as configurable attributes that can be enabled or disabled for added security, testing and to ease the integration effort.

Figure 6 **Security Configuration Settings**

```
<appSettings>
  <add key="acsHostName" value="accesscontrol.windows.net" />
  <add key="serviceNamespace" value="Service Namespace" />
  <add key="trustedAudience"
    value="http://localhost/WCFDataServiceExample/NorthwindService.svc/" />
  <add key="trustedSigningKey" value="Trusted Signing Key" />
  <add key="enableMetadata" value="true" />
  <add key="disableSecurity" value="false"/>
</appSettings>
<system.webServer>
  <validation validateIntegratedModeConfiguration="false" />
  <modules runAllManagedModulesForAllRequests="true">
    <add name="SWTModule" type="Azure.AccessControl.SecurityModule.SWTModule,
      Azure.AccessControl.SecurityModule" preCondition="managedHandler" />
  </modules>
</system.webServer>
```

BEST SELLER



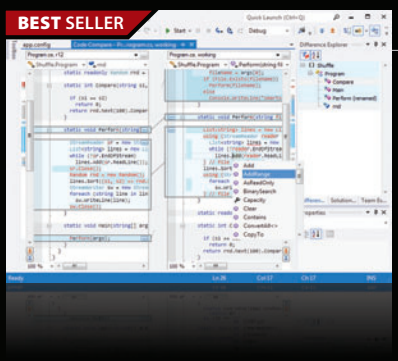
Aspose.Total for .NET | from $2,449.02

ASPOSE Your File Format Experts

Every Aspose .NET component in one package.

- Programmatically manage popular file formats including Word, Excel, PowerPoint and PDF
- Add charting, email, spell checking, barcode creation, OCR, diagramming, imaging, project management and file format management to your .NET applications
- Common uses also include mail merge, adding barcodes to documents, building dynamic Excel reports on the fly and extracting text from PDF files

BEST SELLER



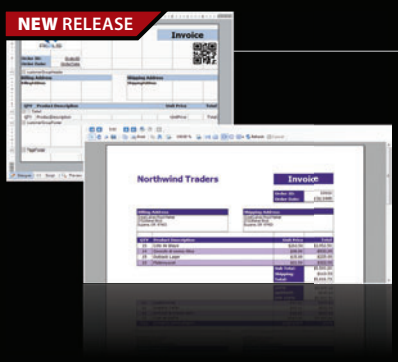
Code Compare Pro | from $48.95

devart

An advanced visual file comparison tool with Visual Studio integration.

- Code oriented comparison, including syntax highlighting, unique structure and lexical comparison algorithms, for the most popular programming languages
- Smooth Visual Studio integration to develop and merge within one environment in the context of current solution, using native IDE editors
- Three-way file merge, folder comparison and synchronization

NEW RELEASE



ActiveReports 7 | from $783.02

ComponentOne a division of GrapeCity

The fast and flexible reporting engine has gotten even better.

- New page layout designer - offers precise design tools for the most complex form reports such as tax forms, insurance forms and investment forms, etc.
- More controls - updated Barcode, Matrix, Calendar and Table controls plus data visualization features
- More customization options - redesigned viewer and designer controls

BEST SELLER

GdPicture.NET | from $3,919.47

GdPicture imaging technologies

A full-featured document-imaging and image processing toolkit for software developers.

- Scan (TWAIN & WIA), process, create, view, edit, annotate, OCR images & PDF files and print documents within your Windows & Web applications
- Read, write and convert vector & raster images in more than 90 formats, including PDF
- Add forms processing, 1D and 2D Barcode recognition features to your programs
- GdPicture SDKs are AnyCPU, thread-safe and 100% royalty-free

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

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

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

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

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

GSA Schedule Contract GS-35F-0188R

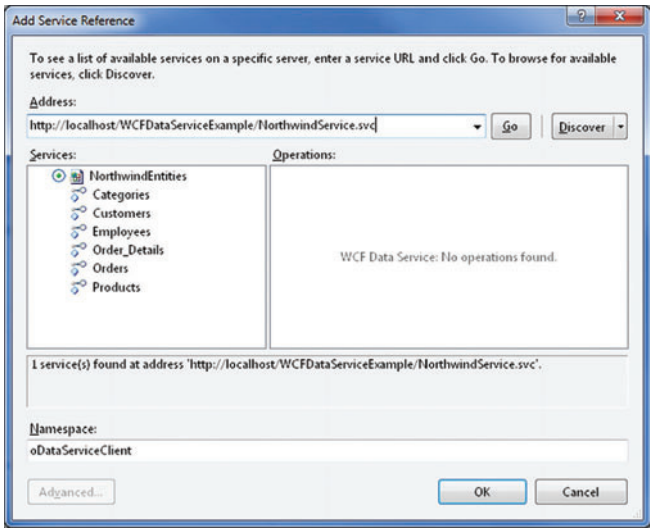


Figure 7 **OData WCF Service with Metadata Exposed**

Figure 5 shows code that intercepts the requests and performs the proper security validation.

Windows Azure SDK

I pulled a class from the Windows Azure SDK to perform token validations for this implementation. The project can be found at bit.ly/utQd3S. After installing the SDK, I copied the file named "tokenvalidator.cs" into a new project. In this particular class, I called the validation method to determine if the user was authorized via the information configured in ACS. To simplify this implementation, I created a custom DLL with only the security mechanism needed. After creating the assembly, all that was needed was a reference to the security DLL with my OData WCF service. The result: a protected and secure implementation.

Implementation of the Secure OData Service

With the additional security enhancement in place, securing the OData WCF service became easy. All that was needed was a reference to the "Azure.AccessControl.SecurityModule" assembly, and to add in the additional configuration settings. Then the security features would be enabled. **Figure 6** shows the security configuration settings.

Depending on the security settings, consumers of my service can be restricted to see only the metadata. This is extremely beneficial because users can still reference the entity objects and properties in code, simplifying the implementation. To disable the metadata, I set the attribute "enableMetadata" to false, so consumers of my service would no longer be able to access the metadata. If consumers of my service were only accessing it via client-side code, I wouldn't enable the metadata because it wouldn't be necessary. The service does appear the same as a normal Web service when the metadata is enabled, but without the ability to consume it without proper authentication and authorization, as shown in **Figure 7**.

This works almost the same as using the Entity Framework directly for the implementation code, with a few minor differences. The key code segment to add is the required token in the request header when sending data to the OData WCF service. I'll explain how the security mechanism works, in essence. First, it checks the header for a valid token and checks if all its components are OK, such as the target audience, token expiration and token value. Next, the request is authorized and the call to the service succeeds. Intercepting this request prior to returning any data to the consumer of the service ensures that the caller of the service had to obtain a valid token prior to being granted access to any data.

Figure 8 **Example Token Request**

```
// Request a token from ACS
using (WebClient client = new WebClient())
{
  client.BaseAddress = string.Format("https://{0}.{1}",
    _accessControlNamespace, _accessControlHostName);
  NameValueCollection values = new NameValueCollection();
  values.Add("wrap_name", wrapUsername);
  values.Add("wrap_password", wrapPassword);
  values.Add("wrap_scope", scope);
  byte[] responseBytes =
    client.UploadValues("WRAPv0.9/", "POST", values);
  response = Encoding.UTF8.GetString(responseBytes);
  Console.WriteLine("\nreceived token from ACS: {0}\n", response);
}
```

At this point—depending on the level of security required on the entity objects—the consumer of the service is able to perform any functionality exposed by the service based on the security settings set. Where the security isn't enabled, the consumer of the service receives an exception that indicates that the action performed isn't allowed.

Unlike traditional Entity Framework code, more logic needs to be implemented prior to calling the Windows Azure-secured OData service. With the HTTP module protecting the service, I need to make sure that I first authenticate to Windows Azure and receive a valid access token prior to calling the OData service. The token received from ACS will be passed through in the request header for every request made to the secure OData service. An example request is shown in **Figure 8**.

Figure 9 **Consuming the OData Secure Service with a Windows Azure Access Token**

```
// First things first: I obtain a token from Windows Azure
_token = GetTokenFromACS(_rootURL + "NorthwindService.svc");
// Now that I have a valid token, I can call the service as needed
Uri uri = new Uri(_rootURL + "NorthwindService.svc/");
try
{
  var northwindEntities = new ODataServiceClient.NorthwindEntities(uri);
  // Add the event handler to send the token in my request header
  northwindEntities.SendingRequest += new
    EventHandler<SendingRequestEventArgs>(OnSendingRequest);
  // Sample selecting data out ...
  var customersFound = from customers in northwindEntities.Customers
    select customers;
  foreach (var customer in customersFound)
  {
    // custom process ...
    // ... <code truncated> ...
    }
    // Add new data in ...
    var category = oDataServiceClient.Category.CreateCategory(0, "New cateogory");
    northwindEntities.AddToCategories(category);
    northwindEntities.SaveChanges();
}
catch (DataServiceRequestException e)
{
  // Trap any data service exceptions such as a security error
  // In the event that the security does not allow an insert,
  // a forbidden error will be returned
  // ...
}
```

Figure 10 **Consuming the OData Secure Service from Client-Side Script**

```
// Parse the entity object into JSON
var jsonEntity = window.JSON.stringify(entityObject);
$.support.cors = true;

// Asynchronous AJAX function to create a Cateogory using OData
$.ajax({
  type: "POST",
  contentType: "application/json; charset=utf-8",
  datatype: "jsonp",
  url: serverUrl + ODATA_ENDPOINT + "/" + odataSetName,
  data: jsonEntity,
  beforeSend: function (XMLHttpRequest) {
  // Specifying this header ensures that the results will be returned as JSON
  XMLHttpRequest.setRequestHeader("Accept", "application/json");
  XMLHttpRequest.setRequestHeader("Authorization", token);
  },
  success: function (data, textStatus, XmlHttpRequest) {
  if (successCallback) {
    successCallback(data.d, textStatus, XmlHttpRequest);
    }
  },
  error: function (XmlHttpRequest, textStatus, errorThrown) {
  if (errorCallback)
    errorCallback(XmlHttpRequest, textStatus, errorThrown);
  else
    errorHandler(XmlHttpRequest, textStatus, errorThrown);
  }
});
```

Once the token is received back from Windows Azure and the user is successfully authenticated and authorized, a token will be returned back from ACS to use for all future requests until the token expires. At this point, implementing the Entity Framework is almost the same as if I were connected to a local database or a database in my network. **Figure 9** shows the consumption of the OData service with an access token.

Implementing the code via client-side script is also just as easy as making an AJAX call to my service endpoint. **Figure 10** shows consuming the OData secure service from client-side script.

A RESTful service provides greater implementation flexibility and is easily consumed via Java or other client-side scripts or APIs. Authentication and a token are still required in order to consume the service, but OData is standard regardless of the platform due to the query syntax. **Figure 11** shows consuming the OData secure service with a Windows Azure access token in Java.

Figure 11 **Java Implementation of Consuming the OData Secure Service with a Windows Azure Access Token**

```
String serviceMethodUrl =
  "http://localhost/WCFDataServiceExample/NorthwindService.svc/Categories?";
GetMethod method = new GetMethod(serviceMethodUrl + "$top=1");

method.addRequestHeader("Authorization", "WRAP access_token=\"" + authToken + "\"");

try
{
  int returnCode = client.executeMethod(method);
  // ... <code truncated> ...
  br = new BufferedReader(new InputStreamReader(method.getResponseBodyAsStream()));

  String readLine;
  while(((readLine = br.readLine()) != null))
  {
    //System.err.println(readLine);
    result += readLine;
  }
}
```

To summarize, I often find the need to expose data in a manner that would require some level of security to prevent unauthorized access. Using ACS supports this need by leveraging a cloud-based federated service to protect not only my OData WCF Data Services, but many other applications as well.

With that said, using WCF Data Services alone would require the implementation of individual data contracts and query interceptors for the data to be exposed. Using the Entity Framework in combination with WCF Data Services provides the ability to leverage database entities as data contracts—and these contracts are provided in a format that's already set up (serializable objects that are accessible via OData). The final piece of the puzzle is to make sure that my OData WCF RESTful services are protected from unauthorized access. Using ACS, OData and the Entity Framework wrapped by WCF RESTful services provides a quick way to expose my data while using standard query syntax with an additional layer of security. ■

Sean Iannuzzi *is a solutions architect for The Agency Inside Harte-Hanks, leveraging best practices for enterprise, system and software solutions. He enjoys learning new technologies and finding ways to leverage technology to help businesses and developers solve problems. He blogs at weblogs.asp.net/seaniannuzzi and you can follow him on Twitter at twitter.com/seaniannuzzi.*

Thanks to the following technical expert for reviewing this article: *Danilo Diaz*

Building an App for Both Windows Phone and iOS

Andrew Whitechapel

There's plenty of documentation about porting apps from iOS to Windows Phone, but in this article, I start from the premise that you want to build a new app from scratch that targets both platforms. I make no value judgments about which platform is better. Instead, I take a practical approach to building the app, and describe the differences and similarities of both platforms encountered on the way.

As a member of the Windows Phone team, I am passionate about the Windows Phone platform—but my key point here is not that one platform is superior to the other, but that the platforms are *different* and require some different programming approaches. Although you can develop iOS apps in C#, using the MonoTouch system, that's a minority environment. For this article, I use standard Xcode and Objective-C for iOS, and Visual Studio and C# for Windows Phone.

This article discusses:

- The UX in both versions of the mobile app
- Core application components
- Fetching and parsing Web data
- Supporting views and services
- Application settings

Technologies discussed:

Windows Phone SDK 7.1, Silverlight for Windows Phone Toolkit, Visual Studio and C#; iOS 5, Xcode 4 and Objective-C

Code download available at:

archive.msdn.microsoft.com/mag201210Phones

Target UX

My aim is to achieve the same UX in both versions of the app while ensuring that each version remains true to the model and philosophy of the target platform. To illustrate what I mean, consider that the Windows Phone version of the app implements the main UI with a vertically scrolling ListBox, whereas the iOS version implements the same UI with a horizontal ScrollViewer. Obviously, these differences are just software—that is, I could build a vertically scrolling list in iOS or a horizontally scrolling list in Windows Phone. Forcing these preferences would be less true to the respective design philosophies, however, and I want to avoid such "unnatural acts."

The app, SeaVan, displays the four land border crossings between Seattle in the United States and Vancouver, British Columbia, in Canada, with the wait times at each of the different crossing lanes. The app fetches the data via HTTP from both U.S. and Canadian government Web sites and refreshes the data either manually via a button or automatically via a timer.

Figure 1 presents the two implementations. One difference you'll notice is that the Windows Phone version is theme-aware and uses the current accent color. In contrast, the iOS version doesn't have a theme or accent-color concept.

The Windows Phone device has a strictly linear page-based navigation model. All significant screen UI is presented as a page, and the user navigates forward and backward through a page stack. You can achieve the same linear navigation on the iPhone, but the iPhone isn't constrained by this model, so you're free to apply whatever screen model you like. In the iOS version of SeaVan, the ancillary screens such as About are modal view controllers. From a technology perspective, these are roughly equivalent to Windows Phone modal popups.

Figure 1 **The Main UI Screen for the SeaVan App on an iPhone and a Windows Phone Device**

Figure 2 presents a schematic of the generalized UI, with internal UI elements in white and external UI elements (launchers/choosers in Windows Phone, shared applications in iOS) in orange. The settings UI (in light green) is an anomaly that I'll describe later in this article.

Another UI difference is that Windows Phone uses an ApplicationBar as a standardized UI element. In SeaVan, this bar is where the user finds buttons to invoke ancillary features in the app—the About page and the Settings page—and to manually refresh the data. There's no direct iOS equivalent to ApplicationBar, so in the iOS version of SeaVan, a simple Toolbar provides the equivalent UX.

Conversely, the iOS version has a PageControl—the black bar at the bottom of the screen, with four positional dot indicators. The user can scroll horizontally through the four border crossings, either by swiping on the content itself or by tapping the PageControl. In Windows Phone SeaVan, there's no PageControl equivalent. Instead, the Windows Phone SeaVan user scrolls through the border crossings by swiping the content directly. One consequence of using a PageControl is that it's easy to configure it so that each page is docked and completely visible. With the Windows Phone scrolling ListBox, there's no standard support for this, so the user can end up with partial views of two border crossings. Both the ApplicationBar and the PageControl are examples of where I haven't attempted to make the UX across the two versions any more uniform than it can be by just using standard behavior.

Architecture Decisions

The use of the Model-View-ViewModel (MVVM) architecture is encouraged in both platforms. One difference is that Visual Studio generates code that includes a reference to the main viewmodel in the application object. Xcode doesn't do this—you're free to plug your viewmodel into your app wherever you choose. In both platforms, it makes sense to plug the viewmodel into the application object.

A more significant difference is the mechanism by which the Model data flows through the viewmodel to the view. In Windows Phone, this is achieved through data binding, which allows you to specify in XAML how UI elements are associated with viewmodel data—and the runtime takes care of actually propagating values. In iOS, while there are third-party libraries that provide similar behavior (based on the Key-Value Observer pattern), there's no data-binding equivalent in the standard iOS libraries. Instead, the app must manually propagate data values between the viewmodel and the view. **Figure 3** illustrates the generalized architecture and components of SeaVan, with viewmodels in pink and views in blue.

Objective-C and C#

A detailed comparison between Objective-C and C# is obviously beyond the scope of a short article, but **Figure 4** provides an approximate mapping of the key constructs.

Core Application Components

To start the SeaVan app, I create a new Single View Application in Xcode and a Windows Phone Application in Visual Studio. Both tools will generate a project with a set of starter files, including classes that represent the application object and the main page or view.

The iOS convention is to use two-letter prefixes in class names, so all the custom SeaVan classes are prefixed with "SV." An iOS app starts with the usual C main method, which creates an app delegate. In SeaVan, this is an instance of the SVAppDelegate class. The app delegate is equivalent to the App object in Windows Phone. I created the project in Xcode with Automatic Reference Counting (ARC) turned on. This adds an @autoreleasepool scope declaration around all code in main, as shown here:

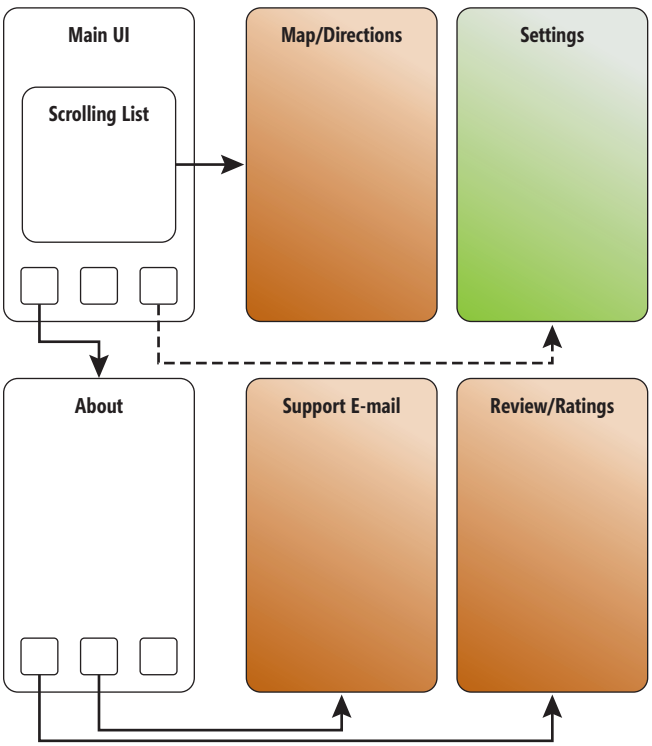


Figure 2 **Generalized Application UI**

```
int main(int argc, char *argv[])
{
  @autoreleasepool {
    return UIApplicationMain(
    argc, argv, nil, NSStringFromClass([SVAppDelegate class]));
  }
}
```

The system will now automatically reference-count the objects I create and automatically release them when the count goes to zero. The @autoreleasepool neatly takes care of most of the usual C/C++ memory management issues and brings the coding experience much closer to C#.

In the interface declaration for SVAppDelegate, I specify it to be a <UIApplicationDelegate>. This means that it responds to standard app delegate messages, such as application:didFinishLaunchingWithOptions. I also declare an SVContentController property. In SeaVan, this corresponds to the standard MainPage class in Windows Phone. The last property is an SVBorderCrossings pointer—this is my main viewmodel, which will hold a collection of SVBorderCrossing items, each representing one border crossing:

```
@interface SVAppDelegate : UIResponder <UIApplicationDelegate>{}
@property SVContentController *contentController;
@property SVBorderCrossings *border;
@end
```

When main starts, the app delegate is initialized and the system sends it the application message with the selector didFinishLaunchingWithOptions. Compare this with Windows Phone, where the logical equivalent would be the application Launching or Activated event handlers. Here, I load an Xcode Interface Builder (XIB) file named SVContent and use it to initialize my main window. The Windows Phone counterpart to a XIB file is a XAML file. XIB files are in fact XML files, although you normally edit them indirectly with the Xcode graphical XIB editor—similar to the graphical XAML editor in Visual Studio. My SVContentController class is associated with the SVContent.xib file, in the same way that the Windows Phone MainPage class is associated with the MainPage.xaml file.

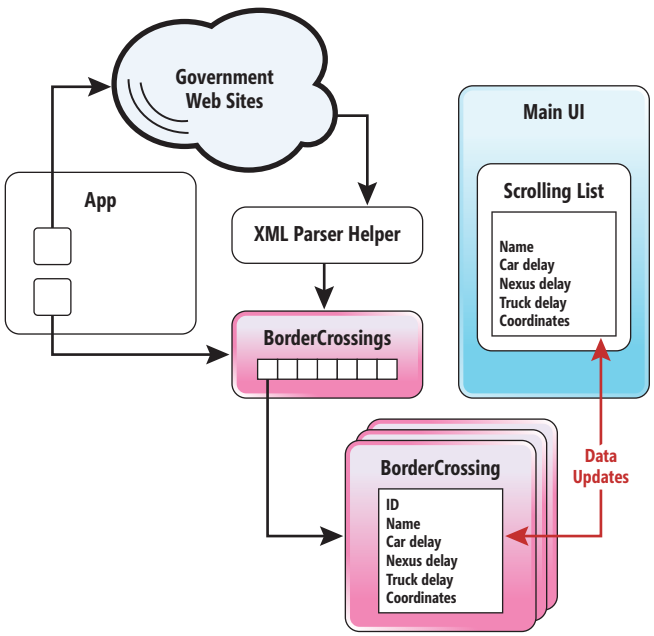


Figure 3 **Generalized SeaVan Architecture**

Finally, I instantiate the SVBorderCrossings viewmodel and invoke its initializer. In iOS, you typically alloc and init in one statement to avoid the potential pitfalls of using uninitialized objects:

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
  [[NSBundle mainBundle] loadNibNamed:@"SVContent" owner:self options:nil];
  [self.window addSubview:self.contentController.view];
  border = [[SVBorderCrossings alloc] init];
  return YES;
}
```

In Windows Phone, the counterpart to the XIB-loading operation is normally done for you behind the scenes. The build generates this part of the code, using your XAML files. For example, if you unhide the hidden files in your Obj folder, in the MainPage.g.cs you'll see the InitializeComponent method, which loads the object's XAML:

```
public partial class MainPage : Microsoft.Phone.Controls.PhoneApplicationPage
{
  public void InitializeComponent()
  {
    System.Windows.Application.LoadComponent(this,
      new System.Uri("/SeaVan;component/MainPage.xaml",
      System.UriKind.Relative));
  }
}
```

The SVContentController class as my main page will host a scroll viewer, which in turn will host four view controllers. Each view controller will eventually be populated with data from one of the four Seattle-Vancouver border crossings. I declare the class to be a <UIScrollViewDelegate> and define three properties: a UIScrollView, a UIPageControl and an NSMutableArray of view controllers. The scrollView and pageControl are both declared as IBOutlet properties, which allows me to connect them to the UI artifacts in the XIB editor. The equivalent in Windows Phone is when an element in XAML is declared with an x:Name, generating the class field. In iOS, you can also connect your XIB UI elements to IBAction properties in your class, allowing you to hook up UI events. The Silverlight equivalent is when you add, say, a Click handler in XAML to hook up the event and provide stub code for the event handler in the class. Interestingly, my SVContentController doesn't subclass any UI class. Instead, it subclasses the base class NSObject. It functions as a UI element in SeaVan because it implements the <UIScrollViewDelegate> protocol—that is, it responds to scrollView messages:

```
@interface SVContentController : NSObject <UIScrollViewDelegate>{}
@property IBOutlet UIScrollView *scrollView;
@property IBOutlet UIPageControl *pageControl;
@property NSMutableArray *viewControllers;
@end
```

In the SVContentController implementation, the first method to be invoked is awakeFromNib (inherited from NSObject). Here, I create the array of SVViewController objects and add each page's view to the scrollView:

```
- (void)awakeFromNib
{
  self.viewControllers = [[NSMutableArray alloc] init];
  for (unsigned i = 0; i < 4; i++)
  {
    SVViewController *controller = [[SVViewController alloc] init];
    [controllers addObject:controller];
    [scrollView addSubview:controller.view];
  }
}
```

Finally, when the user swipes the scrollView or taps the page control, I get the scrollViewDidScroll message. In this method, I switch

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

Code in the Sunshine!
Developers, software architects, programmers and designers will receive hard-hitting and practical .NET Developer training from industry experts and Microsoft insiders. **vslive.com/orlando**

SQL Server LIVE!

TRAINING FOR DBAs AND IT PROS

Bringing SQL Server to Your World
Comprehensive education and knowledge share for IT professionals, developers, DBAs and analytics specialists on SQL Server database management, data warehouse/BI model design, Big Data analytics, performance tuning, troubleshooting and coding against SQL Server. **sqllive360.com**

SharePoint LIVE!

TRAINING FOR COLLABORATION

Build. Develop. Implement. Manage.
Leading-edge knowledge and training for SharePoint administrators, developers, and planners who must customize, deploy and maintain SharePoint Server and SharePoint Foundation to maximize the business value. **splive360.com**

Cloud & Virtualization LIVE!

THE FUTURE OF COMPUTING

Cloud and Virtualization for the Real World
The event for IT Professionals, Systems Administrators, Developers and Consultants to develop their skill sets in evaluating, deploying and optimizing cloud and virtual-based environments. **virtlive360.com**

live360events.com

GOLD SPONSOR axosoft

SUPPORTED BY Microsoft, Microsoft Visual Studio, msdn, Visual Studio MAGAZINE, Redmond MAGAZINE

PRODUCED BY 1105 MEDIA

MEDIA SPONSOR THE CODE PROJECT WWW.CODEPROJECT.COM

Figure 4 **Key Constructs in Objective-C and Their C# Equivalents**

Objective-C	Concept	C# Counterpart
@interface Foo : Bar {}	Class declaration, including inheritance	class Foo : Bar {}
@implementation Foo @end	Class implementation	class Foo : Bar {}
Foo* f = [[Foo alloc] init]	Class instantiation and initialization	Foo f = new Foo();
-(void) doSomething {}	Instance method declaration	void doSomething() {}
+(void) doOther {}	Class method declaration	static void doOther() {}
[myObject doSomething]; *or* myObject.doSomething;	Send a message to (invoke a method on) an object	myObject.doSomething();
[self doSomething]	Send a message to (invoke a method on) the current object	this.doSomething();
-(id)init {}	Initializer (constructor)	Foo() {}
-(id)initWithName:(NSString*)n price:(int)p {}	Initializer (constructor) with parameters	Foo(String n, int p) {}
@property NSString *name;	Property declaration	public String Name { get; set; }
@interface Foo : NSObject <UIAlertViewDelegate>	Foo subclasses NSObject and implements the UIAlertViewDelegate protocol (roughly equivalent to a C# interface)	class Foo : IAnother

the PageControl indicator when more than half of the previous/next page is visible. Then I load the visible page, plus the page on either side of it (to avoid flashes when the user starts scrolling). The last thing I do here is invoke a private method, updateViewFromData, which fetches the viewmodel data and manually sets it into each field in the UI:

```
- (void)scrollViewDidScroll:(UIScrollView *)sender
{
  CGFloat pageWidth = scrollView.frame.size.width;
  int page = floor((scrollView.contentOffset.x - pageWidth / 2) / pageWidth) + 1;
  pageControl.currentPage = page;
  [self loadScrollViewWithPage:page - 1];
  [self loadScrollViewWithPage:page];
  [self loadScrollViewWithPage:page + 1];
  [self updateViewFromData];
}
```

In Windows Phone, the corresponding functionality is implemented in MainPage, declaratively in the XAML. I display the border-crossing times using TextBlock controls within the DataTemplate of a ListBox. The ListBox scrolls each set of data into view automatically, so the Windows Phone SeaVan has no custom code for handling scroll gestures. There's no counterpart for the updateViewFromData method because that operation is taken care of via data binding.

Fetching and Parsing Web Data

As well as functioning as an app delegate, the SVAppDelegate class declares fields and properties to support fetching and parsing the crossing data from the U.S. and Canadian Web sites. I declare two NSURLConnection fields, for the HTTP connections to the two Web sites. I also declare two NSMutableData fields—buffers I'll use to append each chunk of data as it comes in. I update the class to implement the <NSXMLParserDelegate> protocol, so as well as being a standard app delegate, it's also an XML parser delegate. When XML data is received, this class will be called first to parse it. Because I know I'll be dealing with two completely different sets of XML data, I'll immediately hand off the work to one of two child parser delegates. I declare a pair of custom SVXMLParserUs/SVXMLParserCa fields for this. The class also declares a timer for the auto-refresh feature. For each timer event, I'll invoke the refreshData method, as shown in **Figure 5**.

The refreshData method allocates a mutable data buffer for each set of incoming data and establishes the two HTTP connections. I'm using a custom SVURLConnectionWithTag class that subclasses NSURLConnection because the iOS parser delegate model necessitates kicking off both requests from the same object, and all the data will come back into this object. So I need a way to differentiate between the U.S. and Canadian data coming in. To do this, I simply attach a tag to each connection and cache both connections in an NSMutableDictionary. When I initialize each connection, I specify self as the delegate. Whenever a chunk of data is received, the connectionDidReceiveData method is invoked, and I implement this to append the data to the buffer for that tag (see **Figure 6**).

I must also implement connectionDidFinishLoading. When all the data is received (for either of the two connections), I set this app delegate object as the first parser. The parse message is a blocking call, so when it returns, I can invoke updateViewFromData in my content controller to update the UI from the parsed data:

```
- (void)connectionDidFinishLoading:(SVURLConnectionWithTag *)connection
{
  NSXMLParser *parser =
    [[NSXMLParser alloc] initWithData:
  [urlConnectionsByTag objectForKey:connection.tag]];
  [parser setDelegate:self];
  [parser parse];
  [_contentController updateViewFromData];
}
```

Figure 5 **Interface Declaration for SVAppDelegate**

```
@interface SVAppDelegate : UIResponder <UIApplicationDelegate,
NSXMLParserDelegate>
{
  NSURLConnection *connectionUs;
  NSURLConnection *connectionCa;
  NSMutableData *rawDataUs;
  NSMutableData *rawDataCa;
  SVXMLParserUs *xmlParserUs;
  SVXMLParserCa *xmlParserCa;
  NSTimer *timer;
}

@property SVContentController *contentController;
@property SVBorderCrossings *border;
- (void)refreshData;

@end
```

Figure 6 **Setting up the HTTP Connections**

```
static NSString *UrlCa = @"http://apps.cbp.gov/bwt/bwt.xml";
static NSString *UrlUs = @"http://wsdot.wa.gov/traffic/rssfeeds/
CanadianBorderTrafficData/Default.aspx";
NSMutableDictionary *urlConnectionsByTag;

- (void)refreshData
{
  rawDataUs = [[NSMutableData alloc] init];
  NSURL *url = [NSURL URLWithString:UrlUs];
  NSURLRequest *request = [NSURLRequest requestWithURL:url];
  connectionUs =
  [[SVURLConnectionWithTag alloc]
    initWithRequest:request
    delegate:self
    startImmediately:YES
    tag:[NSNumber numberWithInt:ConnectionUs]];

    // ... Code omitted: set up the Canadian connection in the same way
}

- (void)connection:(SVURLConnectionWithTag *)connection
didReceiveData:(NSData *)data
{
  [[urlConnectionsByTag objectForKey:connection.tag] appendData:data];
}
```

In general, there are two types of XML parsers:

- Simple API for XML (SAX) parsers, where your code is notified as the parser walks through the XML tree
- Document Object Model (DOM) parsers, which read the entire document and build up an in-memory representation that you can query for different elements

The default NSXMLParser in iOS is a SAX parser. Third-party DOM parsers are available for use in iOS, but I wanted to compare standard platforms without resorting to third-party libraries. The standard parser works through each element in turn and has no understanding of where the current item fits in the overall XML document. For this reason, the parent parser in SeaVan handles the outermost blocks that it cares about and then hands off to a child delegate parser to handle the next inner block.

In the parser delegate method, I do a simple test to distinguish the U.S. XML from the Canadian XML, instantiate the corresponding child parser and set that child to be the current parser from this point forward. I also set the child's parent parser to self so that the child can return parsing control back to the parent when it gets to the end of the XML that it can handle (see **Figure 7**).

For the equivalent Windows Phone code, I first set up a Web request for the U.S. Web site and for the Canadian Web site. Here I use a WebClient even though an HttpWebRequest is often more appropriate for optimal performance and responsiveness. I set up a handler for the OpenReadCompleted event and then open the request asynchronously:

```
public static void RefreshData()
{
  WebClient webClientUsa = new WebClient();
  webClientUsa.OpenReadCompleted += webClientUs_OpenReadCompleted;
  webClientUsa.OpenReadAsync(new Uri(UrlUs));

  // ... Code omitted: set up the Canadian WebClient in the same way
}
```

In the OpenReadCompleted event handler for each request, I extract the data that has been returned as a Stream object and hand it off to a helper object to parse the XML. Because I have two independent Web requests and two independent OpenReadCompleted event handlers, I don't need to tag the requests or do any testing to identify which request any particular incoming data belongs to. I also don't need to handle each incoming chunk of data to build up the overall XML document. Instead, I can sit back and wait until all the data is received:

Figure 7 **The Parser Delegate Method**

```
- (void)parser:(NSXMLParser *)parser
didStartElement:(NSString *)elementName
  namespaceURI:(NSString *)namespaceURI
  qualifiedName:(NSString *)qName
  attributes:(NSDictionary *)attributeDict
{
  if ([elementName isEqual:@"rss"]) // start of US data
  {
    xmlParserUs = [[SVXMLParserUs alloc] init];
    [xmlParserUs setParentParserDelegate:self];
    [parser setDelegate:xmlParserUs];
  }
  else if ([elementName isEqual:@"border_wait_time"]) // start of Canadian data
  {
    xmlParserCa = [[SVXMLParserCa alloc] init];
    [xmlParserCa setParentParserDelegate:self];
    [parser setDelegate:xmlParserCa];
  }
}
```

```
private static void webClientUs_OpenReadCompleted(object sender,
OpenReadCompletedEventArgs e)
{
  using (Stream result = e.Result)
  {
    CrossingXmlParser.ParseXmlUs(result);
  }
}
```

For parsing the XML, in contrast to iOS, Silverlight includes a DOM parser as standard, represented by the XDocument class. So instead of a hierarchy of parsers, I can use XDocument directly to do all the parsing work:

```
internal static void ParseXmlUs(Stream result)
{
  XDocument xdoc = XDocument.Load(result);
  XElement lastUpdateElement = xdoc.Descendants("last_update").First();

  // ... Etc.
}
```

Supporting Views and Services

In Windows Phone, the App object is static and available to any other component in the application. Similarly, in iOS, one UIApplication

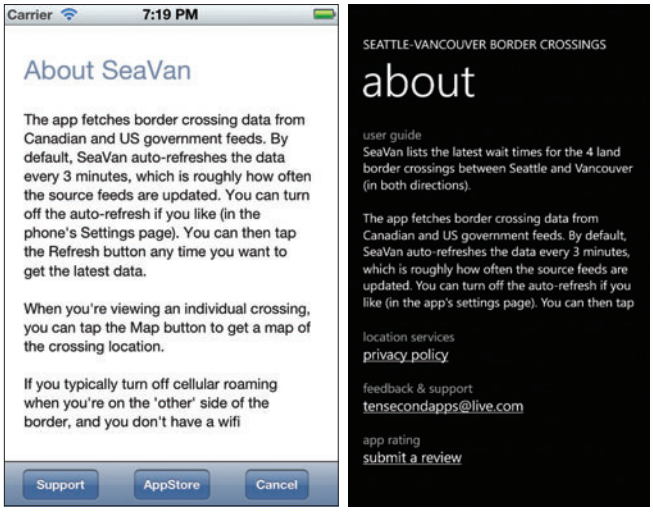


Figure 8 **The SeaVan About Screen in iOS and Windows Phone**

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

delegate type is available in the app. To make things easy, I define a macro that I can use anywhere in the app to get hold of the app delegate, cast appropriately to the specific SVAppDelegate type:

```
#define appDelegate ((SVAppDelegate
*) [[UIApplication sharedApplication]
delegate])
```

This allows me, for example, to invoke the app delegate's refreshData method when the user taps the Refresh button—a button that belongs to my view controller:

```
- (IBAction)refreshClicked:(id)sender
{
  [appDelegate refreshData];
}
```

When the user taps the About button, I want to show an About screen, as shown in **Figure 8**. In iOS, I instantiate an SVAboutViewController, which has an associated XIB, with a scrolling text element for the user guide, as well as three additional buttons in a Toolbar.

To show this view controller, I instantiate it and send the current object (self) a presentModalViewController message:

```
- (IBAction)aboutClicked:(id)sender
{
  SVAboutViewController *aboutView =
    [[SVAboutViewController alloc] init];
  [self presentModalViewController:aboutView animated:YES];
}
```

In the SVAboutViewController class, I implement a Cancel button to dismiss this view controller, making control revert to the invoking view controller:

```
- (IBAction) cancelClicked:(id)sender
{
  [self dismissModalViewControllerAnimated:YES];
}
```

Both platforms offer a standard way for an app to invoke the functionality in built-in apps, such as e-mail, phone and SMS. The key difference is whether control is returned to the app after the built-in functionality returns, which always happens in Windows Phone. In iOS, this happens for some features but not others.

In the SVAboutViewController, when the user taps the Support button, I want to compose an e-mail for the user to send to the development team. The MFMailComposeViewController—again presented as a modal view—works well for this purpose. This standard view controller also implements a Cancel button, which does exactly the same work to dismiss itself and revert control to its invoking view:

```
- (IBAction)supportClicked:(id)sender
{
  if ([MFMailComposeViewController canSendMail])
  {
    MFMailComposeViewController *mailComposer =
      [[MFMailComposeViewController alloc] init];
    [mailComposer setToRecipients:
      [NSArray arrayWithObject:@"tensecondapps@live.com"]];
    [mailComposer setSubject:@"Feedback for SeaVan"];
    [self presentModalViewController:mailComposer animated:YES];

  }
```

The standard way to get map directions in iOS is to invoke Google maps. The downside to this approach is that it takes the user out to the Safari shared application (built-in app), and there's no way to programmatically return control to the app. I want to minimize the places where the user leaves the app, so instead of directions, I present a map of the target border crossing by using a custom SVMapViewController that hosts a standard MKMapView control:

```
- (IBAction)mapClicked:(id)sender
{
  SVBorderCrossing *crossing =
    [appDelegate.border.crossings
    objectAtIndex:parentController.pageControl.currentPage];
  CLLocationCoordinate2D target = crossing.coordinatesUs;
  SVMapViewController *mapView =
    [[SVMapViewController alloc]
    initWithCoordinate:target title:crossing.portName];
  [self presentModalViewController:mapView animated:YES];
}
```

To allow the user to enter a review, I can compose a link to the app in the iTunes App Store. (The nine-digit ID in the following code is the app's App Store ID.) I then pass this to the Safari browser (a shared application). I have no option here but to leave the app:

```
- (IBAction)appStoreClicked:(id)sender
{
  NSString *appStoreURL =
    @"http://itunes.apple.com/us/app/id123456789?mt=8";
  [[UIApplication sharedApplication]
    openURL:[NSURL URLWithString:appStoreURL]];
}
```

The Windows Phone equivalent of the About button is a button on the ApplicationBar. When the user taps this button, I invoke the NavigationService to navigate to the AboutPage:

```
private void appBarAbout_Click(object sender, EventArgs e)
{
  NavigationService.Navigate(new Uri("/AboutPage.xaml", UriKind.Relative));
}
```

Just as in the iOS version, the AboutPage presents a simple user guide in scrolling text. There's no Cancel button because the user can tap the hardware Back button to navigate back from this page. Instead of the Support and App Store buttons, I have HyperlinkButton controls. For the support e-mail, I can implement the behavior declaratively by using a NavigateUri that specifies the mailto: protocol. This is sufficient to invoke the EmailComposeTask:

```
<HyperlinkButton
  Content="tensecondapps@live.com" Margin="-12,0,0,0"
HorizontalAlignment="Left"
  NavigateUri="mailto:tensecondapps@live.com" TargetName="_blank" />
```

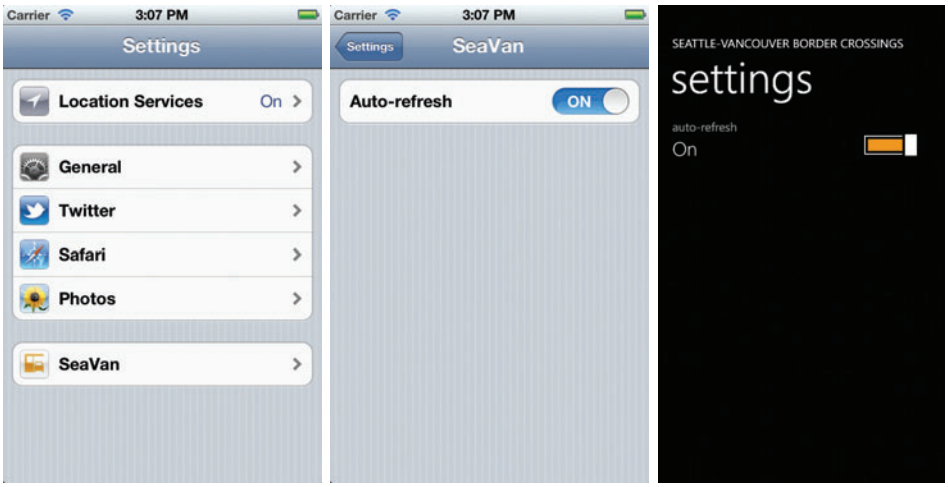


Figure 9 **Standard Settings and SeaVan-Specific Settings on iOS and the Windows Phone Settings Page**

Kendo UI

THE ART OF WEB DEVELOPMENT

Everything You Want

JavaScript Grids and Charts with awesome themes

Real applications need tools ready for business. Kendo UI includes high-performance and feature-rich UI widgets like Grid, Charts, TreeView, and ComboBox. Use JavaScript, HTML5, and Kendo UI to quickly build business applications that look amazing in any browser with professionally designed CSS3 themes. If you want it, Kendo UI has got it.

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

I set up the Review link with a Click handler in code, and then I invoke the MarketplaceReviewTask launcher:

```
private void ratingLink_Click(object sender, RoutedEventArgs e)
{
  MarketplaceReviewTask reviewTask = new MarketplaceReviewTask();
  reviewTask.Show();
}
```

Back in the MainPage, rather than offer a separate button for the Map/Directions feature, I implement the SelectionChanged event on the ListBox so that the user can tap on the content to invoke this feature. This approach is in keeping with Windows Store apps, in which the user should interact directly with the content rather than indirectly via chrome elements. In this handler, I fire up a BingMapsDirectionsTask launcher:

```
private void CrossingsList_SelectionChanged(
  object sender, SelectionChangedEventArgs e)
{
  BorderCrossing crossing = (BorderCrossing)CrossingsList.SelectedItem;
  BingMapsDirectionsTask directions = new BingMapsDirectionsTask();
  directions.End =
    new LabeledMapLocation(crossing.PortName, crossing.Coordinates);
  directions.Show();
}
```

Application Settings

On the iOS platform, app preferences are managed centrally by the built-in Settings app, which provides a UI for users to edit settings for both built-in and third-party apps. **Figure 9** shows the main Settings UI and the specific SeaVan settings view in iOS, and the Windows Phone settings page. There's just one setting for SeaVan—a toggle for the auto-refresh feature.

To incorporate settings within an app, I use Xcode to create a special type of resource known as a settings bundle. Then I configure the settings values by using the Xcode settings editor—no code required.

In the application method, shown in **Figure 10**, I make sure the settings are in sync and then fetch the current value from the store. If the auto-refresh setting value is True, I start the timer. The APIs support both getting and setting the values within the app, so I could optionally provide a settings view in the app in addition to the app's view in the Settings app.

In Windows Phone, I can't add the app settings to the global settings app. Instead, I provide my own settings UI within the app. In SeaVan,

Figure 10 **The Application Method**

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
  NSUserDefaults *defaults =
    [NSUserDefaults standardUserDefaults];
  [defaults synchronize];
  boolean_t isAutoRefreshOn =
    [defaults boolForKey:@"autorefresh"];
  if (isAutoRefreshOn)
  {
    [timer invalidate];
    timer =
      [NSTimer scheduledTimerWithTimeInterval:kRefreshIntervalInSeconds
        target:self
        selector:@selector(onTimer)
        userInfo:nil
        repeats:YES];
  }

  // ... Code omitted for brevity

  return YES;
}
```

Version Notes and Sample App

Platform versions:

- Windows Phone SDK 7.1, and the Silverlight for Windows Phone Toolkit
- iOS 5 and Xcode 4

SeaVan will be published to both the Windows Phone Marketplace and the iTunes App Store.

as with the AboutPage, the SettingsPage is simply another page. I provide a button on the ApplicationBar to navigate to this page:

```
private void appBarSettings_Click(object sender, EventArgs e)
{
  NavigationService.Navigate(new Uri("/SettingsPage.xaml", UriKind.Relative));
}
```

In the SettingsPage.xaml, I define a ToggleSwitch for the auto-refresh feature:

```
<StackPanel x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
  <toolkit:ToggleSwitch
    x:Name="autoRefreshSetting" Header="auto-refresh"
    IsChecked="{Binding Source={StaticResource appSettings},
    Path=AutoRefreshSetting, Mode=TwoWay}"/>
</StackPanel>
```

I have no choice but to provide settings behavior within the app, but I can turn this to my advantage and implement an AppSettings viewmodel for it and hook it up to the view via data binding, just as with any other data model. In the MainPage class, I start the timer based off the value of the setting:

```
protected override void OnNavigatedTo(NavigationEventArgs e)
{
  if (App.AppSettings.AutoRefreshSetting)
  {
    timer.Tick += timer_Tick;
    timer.Start();
  }
}
```

Not That Difficult

Building one app that targets both iOS and Windows Phone isn't that difficult: the similarities are greater than the differences. Both use MVVM with an app object and one or more page/view objects, and the UI classes are associated with XML (XAML or XIB), which you edit with a graphical editor. In iOS, you send a message to an object, while in Windows Phone you invoke a method on an object. But the difference here is almost academic, and you can even use dot notation in iOS if you don't like the "[message]" notation. Both platforms have event/delegate mechanisms, instance and static methods, private and public members, and properties with get and set accessors. On both platforms, you can invoke built-in app functionality and support user settings. Obviously, you have to maintain two codebases—but your app's architecture, major component design and UX can be held consistent across the platforms. Try it—you'll be pleasantly surprised! ■

ANDREW WHITECHAPEL *has been a developer for more than 20 years and currently works as a program manager on the Windows Phone team, responsible for core pieces of the application platform. His new book is "Windows Phone 7 Development Internals" (Microsoft Press, 2012).*

THANKS to the following technical experts for reviewing this article: *Chung Webster and Jeff Wilcox*

Tombstoning and the Zen of Async for Windows Phone

Benjamin Day

Have you ever written an application and almost as soon as you're finished, wish you hadn't written it the way you did? It's that gut feeling that something isn't quite right with the architecture. Changes that should be simple feel next to impossible, or at least take much longer than they should. And then there are the bugs. Oh, there are bugs! You're a decent programmer. How'd you manage to write something with so many bugs?

Sound familiar? Well, it happened to me when I wrote my first Windows Phone application, NPR Listener. NPR Listener talks to National Public Radio Web services (npr.org/api/index.php) to get the list of available stories for its programs, and then lets users listen to those stories on their Windows Phone devices. When I first wrote it, I'd been doing lots of Silverlight development and I was very happy at how well my knowledge and skills ported to Windows Phone. I got the first version done fairly quickly and submitted it to the Marketplace certification process. The entire time I was thinking, "Well, that was easy." And then I failed certification. Here's the case that failed:

Step 1: Run your application.

Step 2: Press the Start button to go to the main page of your phone.

Step 3: Press the Back button to return to your application.

This article discusses:

- Tombstoning and the Windows Phone 7 navigation back stack
- Using isolated storage for caching and simplified network operations
- Using isolated storage for continuous tombstoning

Technologies discussed:

Windows Phone 7.5

Code download available at:

v

When you press the Back button, your application should resume without error and, ideally, should put the user right back at the screen where he exited your application. In my case, the tester navigated to a National Public Radio program (such as "All Things Considered"), clicked into one of the current stories and then pressed the Start button to go to the home screen on the device. When the tester pressed the Back button to return to my app, the application came back up, and it was a festival of NullReferenceExceptions. Not good.

Now, I'll let you know a bit about how I design my XAML-based applications. For me, it's all about the Model-View-ViewModel pattern, and I aim for a nearly fanatical separation between the XAML pages and the logic of my app. If there's going to be any code in the codebehinds (*.xaml.cs) of my pages, there had better

Figure 1 **Simple Tombstone Implementation in App.xaml.cs**

```
// Code to execute when the application is deactivated (sent to background).
// This code will not execute when the application is closing.
private void Application_Deactivated(object sender, DeactivatedEventArgs e)
{
  // Tombstone your application.

  IDictionary<string, object> stateCollection =
    PhoneApplicationService.Current.State;

  stateCollection.Add("VALUE_1", "the value");
}

// Code to execute when the application is activated
// (brought to foreground).
// This code will not execute when the application is first launched.
private void Application_Activated(object sender, ActivatedEventArgs e)
{
  // Un-tombstone your application.

  IDictionary<string, object> stateCollection =
    PhoneApplicationService.Current.State;

  var value = stateCollection["VALUE_1"];
}
```

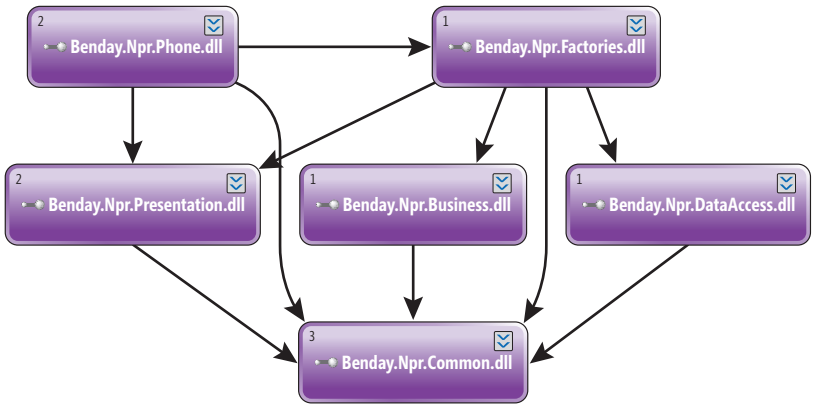


Figure 2 **The Structure of the Application**

be an extremely good reason. A lot of this is driven by my near-pathological need for unit testability. Unit tests are vital because they help you know when your application is working and, more importantly, they make it easy to refactor your code and change how the application works.

So, if I'm so fanatical about unit tests, why did I get all those NullReferenceExceptions? The problem is that I wrote my Windows Phone application like a Silverlight application. Sure, Windows Phone is Silverlight, but the lifecycle of a Windows Phone app and a Silverlight app are completely different. In Silverlight, the user opens the application, interacts with it until she's done and then closes the app. In Windows Phone, in contrast, the user opens the application, works with it and bounces back and forth—into the OS or any other application—whenever she wants. When she moves away from your application, the application is deactivated, or "tombstoned." When your app has been tombstoned, it's no longer running, but its navigation "back stack"—the pages in the application in the order they were visited—is still available on the device.

You might have noticed on your Windows Phone device that you can navigate through a number of applications and then press the Back button repeatedly to go back through those applications in reverse order. That's the navigation back stack in action, and each time you go into a different application, that application is reactivated from persisted tombstone data. When your application is going to be tombstoned, it gets a notification from the OS that it's about to be deactivated and should save its application state so it can be reactivated later. **Figure 1** shows some simple code for activating and deactivating your application in App.xaml.cs.

My NullReferenceException problem was caused by a complete lack of planning—and coding—to handle those tombstoning events. That, plus my comprehensive, rich and complex ViewModel implementation, was a recipe for disaster. Think about what happens when a user clicks the Back button to reenter your application. That user doesn't end up at some start page but instead lands at the last page he visited in your app. In the case of the Windows Phone tester, when the user reactivated my application, she entered the application in the middle and the UI assumed that the ViewModel was populated and could support that screen. Because the ViewModel wasn't responding to tombstone events, almost every object reference was null. Oops. I didn't unit test that case, did I? (Kaboom!)

The lesson here is that you need to plan your UI and ViewModels for navigating both forward and backward.

Adding Tombstoning After the Fact

Figure 2 shows the structure of my original application. In order to make my application pass certification, I needed to handle that Start/Back button case. I could either implement tombstoning in the Windows Phone project (Benday.Npr.Phone) or I could force it into my ViewModel (Benday.Npr.Presentation). Both involved some uncomfortable architectural compromises. If I added the logic to the Benday.Npr.Phone project, my UI would know too much about how my ViewModel works. If I added the logic to

Figure 3 **Code in StoryListViewModelSerializer.cs to Turn IStoryListViewModel into XML**

```
private void Serialize(IStoryListViewModel fromValue)
{
  var document = XmlUtility.StringToXDocument("<stories />");

  WriteToDocument(document, fromValue);

  // Write the XML to the tombstone dictionary.
  SetStateValue(SERIALIZATION_KEY_STORY_LIST, document.ToString());
}

private void WriteToDocument(System.Xml.Linq.XDocument document,
  IStoryListViewModel storyList)
{
  var root = document.Root;

  root.SetElementValue("Id", storyList.Id);
  root.SetElementValue("Title", storyList.Title);
  root.SetElementValue("UrlToHtml", storyList.UrlToHtml);

  var storySerializer = new StoryViewModelSerializer();

  foreach (var fromValue in storyList.Stories)
  {
    root.Add(storySerializer.SerializeToElement(fromValue));
  }
}
```

Figure 4 **Triggering the ViewModel Serializers in App.xaml.cs**

```
private void Application_Deactivated(object sender, DeactivatedEventArgs e)
{
  ViewModelSerializerBase.ClearState();

  if (IsDisplayingStory() == true)
  {
    new StoryListViewModelSerializer().Serialize();
    new StoryViewModelSerializer().Serialize();

    ViewModelSerializerBase.SetResumeActionToStory();
  }
  else if (IsDisplayingProgram() == true)
  {
    new StoryListViewModelSerializer().Serialize();
    new ProgramViewModelSerializer().Serialize();

    ViewModelSerializerBase.SetResumeActionToProgram();
  }
  else if (IsDisplayingHourlyNews() == true)
  {
    new StoryListViewModelSerializer().Serialize();

    ViewModelSerializerBase.SetResumeActionToHourlyNews();
  }
}
```

the ViewModel project, I'd need to add a reference from Benday.Npr.Presentation to Microsoft.Phone.dll to get access to the tombstone value dictionary (PhoneApplicationService.Current.State) in the Microsoft.Phone.Shell namespace. That would pollute my ViewModel project with unnecessary implementation details and would be a violation of the Separation of Concerns (SoC) principle.

My eventual choice was to put the logic into the Phone project but also create some classes that know how to serialize my ViewModel into an XML string that I could put into the tombstone value dictionary. This approach allowed me to avoid the reference from the Presentation project to Microsoft.Phone.Shell while still giving me clean code that honored the Single Responsibility Principle. I named these classes *ViewModelSerializer. **Figure 3** shows some of the code required to turn an instance of StoryListViewModel into XML.

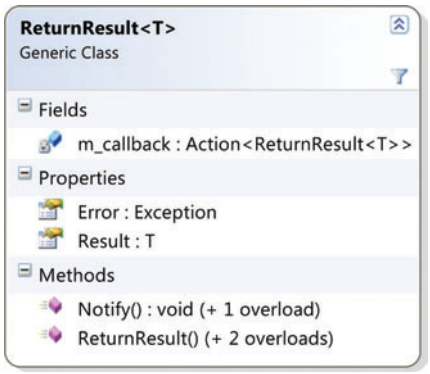


Figure 5 **ReturnResult<T>**

Once I had these serializers written, I needed to add logic to App.xaml.cs to trigger this serialization based on the screen currently being displayed (see **Figure 4**).

I eventually got it working and got the application certified but, unfortunately, the code was slow, ugly, brittle and buggy. What I should've done was design my ViewModel so it had less state that needed to be saved, and then build it so it would persist itself as it ran, rather than having to do one giant tombstoning event at the end. How would I do that?

The Control Freak School of Asynchronous Programming

So, here's a question for you: Do you have "control freak" tendencies? Do you have trouble letting go? Do you choose to ignore obvious truths and, through sheer force of will, solve problems in ways that ignore the reality that's clear as day and staring you directly in the eyes? Yup … that's how I handled asynchronous calls in the first version of NPR Listener. Specifically, that's how I approached the asynchronous networking in the first version of the application.

In Silverlight, all network calls must be asynchronous. Your code initiates a network call and immediately returns. The result (or an exception) is delivered sometime later via an asynchronous callback. This means that networking logic always consists of two pieces—the outgoing call and the return call. This structure has consequences and it's a dirty little secret in Silverlight that any method that relies on the results of a network call can't return a value and must return void. This has a side effect: Any method that calls *another* method that relies on the results of a network call must also return void. As you might imagine, this can be absolutely brutal for layered architectures because the traditional implementations of *n*-tier design patterns, such as Service Layer, Adapter and Repository, rely heavily on return values from method calls.

My solution is a class called ReturnResult<T> (shown in **Figure 5**), which serves as the glue between the method that requests the network call and the method that handles the results of the call and provides a way for your code to return useful values. **Figure 6** shows some Repository Pattern logic that makes a call to a Windows Communication Foundation (WCF) service and then returns a populated instance of IPerson. Using that code, you can call LoadById(ReturnResult<IPerson>, int) and eventually receive the populated instance of IPerson when client_LoadByIdCompleted(object, LoadByIdCompletedEventArgs) calls one of the Notify methods. It basically allows you to create code that's similar to what you'd have if you could use return values. (For more information about ReturnResult<T>, please see bit.ly/Q6dqIv.)

When I finished writing the first version of NPR Listener, I quickly figured out that the application was slow (or at least appeared to be slow) because I didn't do any caching. What I really needed in the app was a way to call an NPR Web service, get a list of stories for a given program and then cache that data so I didn't have to go back to the service every time I needed to draw that screen. Adding that functionality, however, was fairly difficult because I was trying to pretend that the async calls didn't exist. Basically, by being a control freak and trying to deny the essentially asynchronous structure of my application, I was limiting my options. I was fighting the platform and therefore contorting my application architecture.

Figure 6 **Using ReturnResult<T> to Start a Network Call and Return a Value from the Completed Event**

```
public void LoadById(ReturnResult<IPerson> callback, int id)
{
  // Create an instance of a WCF service proxy.
  var client = new PersonService.PersonServiceClient();

  // Subscribe to the "completed" event for the service method.
  client.LoadByIdCompleted +=
    new EventHandler<PersonService.LoadByIdCompletedEventArgs>(
      client_LoadByIdCompleted);

  // Call the service method.
  client.LoadByIdAsync(id, callback);
}

void client_LoadByIdCompleted(object sender,
  PersonService.LoadByIdCompletedEventArgs e)
{
  var callback = e.UserState as ReturnResult<IPerson>;

  if (e.Error != null)
  {
    // Pass the WCF exception to the original caller.
    callback.Notify(e.Error);
  }
  else
  {
    PersonService.PersonDto personReturnedByService = e.Result;

    var returnValue = new Person();

    var adapter = new PersonModelToServiceDtoAdapter();

    adapter.Adapt(personReturnedByService, returnValue);

    // Pass the populated model to the original caller.
    callback.Notify(returnValue);
  }
}
```

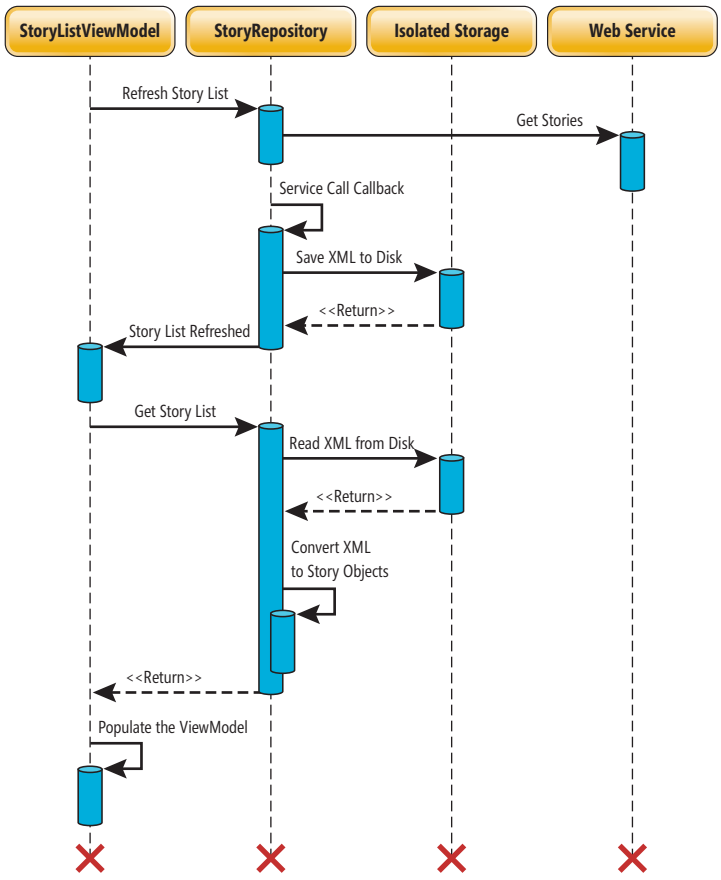


Figure 7 **Sequence Diagram for Refreshing and Loading the Story List**

In a synchronous application, things start to happen in the UI and the flow of control passes through the layers of the app, returning data as the stack unwinds. Everything happens inside a single call stack where the work is initiated, data is processed and a return value is returned back up the stack. In an asynchronous application, the process is more like four calls that are all loosely connected: the UI requests that something happens; some processing may or may not happen; if the processing happens and the UI has subscribed to the event, the processing notifies the UI that an action completed; and the UI updates the display with the data from the asynchronous action.

I can already picture myself lecturing some young whippersnapper about how hard we had it in the days before async and await. "In my day, we had to manage our own asynchronous networking logic and callbacks. It was brutal, and we liked it! Now get off my lawn!" Well, in truth, we didn't like it that way. It *was* brutal.

Here's another lesson: Fighting the underlying architecture of the platform will always cause you problems.

Rewriting the Application Using Isolated Storage

I first wrote the application for Windows Phone 7 and did only a minor update for Windows Phone 7.1. In an application whose entire mission is to stream audio, it had always been a disappointment that users couldn't listen to audio while browsing other applications. When Windows Phone 7.5 came out, I wanted to take advantage of the new background-streaming features. I also wanted to speed up the application and eliminate a lot of unnecessary Web service calls by adding some kind of local data caching. As I started thinking about implementing these features, though, the limitations and the brittleness of my tombstoning, ViewModel and async implementation became more and more apparent. It was time to fix my previous mistakes and completely rewrite the application.

Having learned my lessons in the previous version of the application, I decided I was going to start by designing for "tombstone-ability" and also completely embrace the asynchronous nature of the application. Because I wanted to add local data caching, I started looking into using isolated storage. Isolated storage is a location on your device where your app can read and write data. Working with it is similar to working with the file system in any ordinary .NET application.

Isolated Storage for Caching and Simplified Network Operations

A huge upside of isolated storage is that these calls, unlike network calls, don't have to be asynchronous. That means I can use a more-conventional architecture that relies on return values. Once I figured this out, I started thinking about how to separate the operations that have to be asynchronous from the ones that can be synchronous. Network calls have to be async. Isolated storage calls can be synchronous. So what if I always write the results of the network calls to isolated storage before I do any parsing? This lets me synchronously load data and gives me a cheap and easy way to do local data caching. Isolated storage helps me solve two problems at once.

I started by reworking how I do my network calls, embracing the fact that they're a series of loosely associated steps instead of just one big synchronous step. For example, when I want to get a list of stories for a given NPR program, here's what I do (see **Figure 7**):

1. The ViewModel subscribes to a StoryListRefreshed event on the StoryRepository.
2. The ViewModel calls the StoryRepository to request a refresh of the story list for the current program. This call completes immediately and returns void.
3. The StoryRepository issues an asynchronous network call to an NPR REST Web service to get the list of stories for the program.

Figure 8 **StoryRepository Logic to Save Current Story Id**

```
public void SaveCurrentStoryId(string currentId)
{
  var doc = XmlUtility.StringToXDocument("<info />");

  if (currentId == null)
  {
    currentId = String.Empty;
  }
  else
  {
    currentId = currentId.Trim();
  }

  XmlUtility.SetChildElement(doc.Root, "CurrentStoryId", currentId);

  DataAccessUtility.SaveFile(DataAccessConstants.FilenameStoryInformation, doc);
}
```

Figure 9 **StoryListViewModel Saves the Current Story Id When the Value Changes**

```
void m_Stories_OnItemSelected(object sender, EventArgs e)
{
  HandleStorySelected();
}

private void HandleStorySelected()
{
  if (Stories.SelectedItem == null)
  {
    CurrentStoryId = null;
    StoryRepositoryInstance.SaveCurrentStoryId(null);
  }
  else
  {
    CurrentStoryId = Stories.SelectedItem.Id;
    StoryRepositoryInstance.SaveCurrentStoryId(CurrentStoryId);
  }
}
```

4. At some point, the callback method is triggered and the StoryRepository now has access to the data from the service. The data comes back from the service as XML and, rather than turning this into populated objects that get returned to the ViewModel, the StoryRepository immediately writes the XML to isolated storage.
5. The StoryRepository triggers a StoryListRefreshed event.
6. The ViewModel receives the StoryListRefreshed event and calls GetStories to get the latest list of stories. GetStories reads the cached story list XML from isolated storage, converts it into objects that the ViewModel needs and returns the populated objects. This method can return populated objects because it's a synchronous call that reads from isolated storage.

The important point here is that the RefreshStories method doesn't return any data. It just requests the refresh of the cached story data. The GetStories method takes the currently cached XML data and converts it into IStory objects. Because GetStories doesn't have to call any services, it's extremely fast, so the Story List screen populates quickly and the application seems much faster than the first version. If there's no cached data, GetStories simply returns an empty list of IStory objects. Here's the IStoryRepository interface:

```
public interface IStoryRepository
{
  event EventHandler<StoryListRefreshedEventArgs> StoryListRefreshed;

  IStoryList GetStories(string programId);
  void RefreshStories(string programId);

    ...
}
```

An additional point about hiding this logic behind an interface is that it makes for clean code in the ViewModels and decouples the development effort of the ViewModels from the storage and service logic. This separation makes the code much easier to unit test and much easier to maintain.

Isolated Storage for Continuous Tombstoning

My tombstoning implementation in the first version of the application took the ViewModels and converted them into XML that was stored in the phone's tombstone value dictionary, PhoneApplicationService.Current.State. I liked the XML idea but didn't like that persistence of the ViewModel was the responsibility of the phone app's UI tier rather than of the ViewModel tier itself. I also didn't like that the UI tier waited until the tombstone Deactivate event to persist my entire set of ViewModels. When the app is running, only a handful of values actually need to be persisted, and they change very gradually as the user moves from screen to screen. Why not write the values to isolated storage as the user navigates through the app? That way the app is always ready to be deactivated and tombstoning isn't a big deal.

Moreover, instead of persisting the entire state of the application, why not save only the currently selected value on each page? The data is cached locally so it should be on the device already, and I can easily reload the data from the cache without changing the logic of the application. This decreases the number of values that have to be persisted from the hundreds in version 1 to maybe four or five in version 2. That's a lot less data to worry about, and everything is much simpler.

The logic for all the persistence code for reading and writing to or from isolated storage is encapsulated in a series of Repository objects. For information related to Story objects, there will be a corresponding StoryRepository class. **Figure 8** shows the code for taking a story Id, turning it into an XML document and saving it to isolated storage.

Wrapping the persistence logic inside a Repository object keeps the storage and retrieval logic separated from any ViewModel logic and hides the implementation details from the ViewModel classes. **Figure 9** shows the code in the StoryListViewModel class for saving the current story Id when the story selection changes.

And here's the StoryListViewModel Load method, which reverses process when the StoryListViewModel needs to repopulate itself from disk:

```
public void Load()
{
  // Get the current story Id.
  CurrentStoryId = StoryRepositoryInstance.GetCurrentStoryId();

  ...

  var stories = StoryRepositoryInstance.GetStories(CurrentProgramId);
  Populate(stories);
}
```

Plan Ahead

In this article, I've walked you through some of the architectural decisions and mistakes that I made in my first Windows Phone application, NPR Listener. Remember to plan for tombstoning and to embrace—rather than fight—async in your Windows Phone applications. If you'd like to look at the code for both the before and after versions of NPR Listener, you can download it from archive.msdn.microsoft.com/mag201209WP7. ■

BENJAMIN DAY *is a consultant and trainer specializing in software development best practices using Microsoft development tools with an emphasis on Visual Studio Team Foundation Server, Scrum and Windows Azure. He's a Microsoft Visual Studio ALM MVP, a certified Scrum trainer via Scrum.org, and a speaker at conferences such as TechEd, DevTeach and Visual Studio Live! When not developing software, Day has been known to go running and kayaking in order to balance out his love of cheese, cured meats and champagne. He can be contacted via benday.com.*

THANKS to the following technical experts for reviewing this article: *Jerri Chiu and David Starr*

TEAM FOUNDATION SERVER 2012 HOSTING

30 DAY FREE TRIAL + NO SETUP FEES

WWW.DISCOUNTASP.NET/TFS/MSDN

As a Microsoft Gold Hosting Partner and Microsoft Visual Studio Partner, we offer all the tools development teams require to effectively manage their software development projects without the aggravation or expense of running TFS on their own.

discountASP.net
Team Foundation Server Hosting

Microsoft Partner
Gold Hosting

Microsoft Visual Studio
PARTNER

TFS HOSTING FEATURES

$20/mo per user
Monthly Billing
Daily Backups
Unlimited Projects
5gb of Disk Space
Source Control
Work Item Tracking
Team Web Access
Custom Process Template
Secure Access via HTTPS
Visual Studio 2012/2010 Integration

TFS HOSTING ADDONS & MIGRATION SERVICES

- TFS Build Server
- VSS to TFS Migration
- On-premise TFS 2012 to Hosted TFS Migration

Testing Math Functions in Microsoft Cloud Numerics

Stuart Brorson, Alan Edelman and Ben Moskowitz

Suppose you need to perform a mathematical calculation. For example, suppose you need to know the sine of 34°. What do you do? You probably turn to a calculator, computer or some other smart device. On your device, you type in "sin(34)" and you get an answer, often with 16 decimal digits of precision. But how do you know your answer is correct?

We're so accustomed to getting mathematical answers from our electronic gizmos that nobody thinks about whether the answers are correct! Just about everybody takes it for granted that our machines give us the right answers. However, for a small set of software quality engineers, correctness can't be taken for granted; the job is all about getting the right answer. This article explains how the math functions in the new Microsoft Cloud Numerics math library are tested.

The article discusses testing performed on Microsoft Cloud Numerics, which has been released as a lab.

This article discusses:

- Computing accurate functions
- Testing function accuracy
- Determining testing tolerance
- Different kinds of tests
- Validating the approach

Technologies discussed:

Microsoft Cloud Numerics

Most scientific and engineering computations use floating-point arithmetic. The IEEE standardized the basic workings of floating-point math in 1985. A key feature of the standard is that it acknowledges not all numbers can be represented on a computer. While the set of real numbers is infinite and uncountable, the set of IEEE floating-point numbers is necessarily finite and countable because the numeric representation of floating-point numbers uses a fixed number of bits. The construction of IEEE floating-point numbers implies that they're also rational, so commonly used numbers such as π aren't expressed exactly, and operations using them, such as *sin(x)*, aren't exact, either.

Moreover, unlike with integers, the spacing between IEEE floating-point numbers is locally uniform, but it increases logarithmically with the magnitude of the number itself. This logarithmic aspect is depicted in **Figure 1**, which shows schematically the locations of uniform chunks of IEEE floating-point numbers on the real number line. In the figure, valid chunks of floating-point numbers (embedded in the real number line) are indicated by vertical lines. Note that the distance between valid floating-point numbers increases logarithmically as the magnitude of *x* increases. The distance between two adjacent floating-point numbers is often called the Unit of Least Precision or Unit in Last Place (ULP), and is provided as a built-in function called *eps(x)* in many common math programs. A consequence of the variation in spacing is that a small number close to 0 has no effect when added to a relatively larger number such as 1.

Besides codifying formats for floating-point numbers, the IEEE standard provides strict guidance about how accurate the returns from the most fundamental math operations must be. For example, the IEEE standard specifies that the returns from the four arithmetic operations (+, -, * and /) must be "best rounded," meaning that the answer returned must be *closest* to the "mathematically correct" result. Although this requirement was originally met with some resistance, returning the best-rounded result now takes place in hardware, showing how commonplace it has become. More interestingly, the IEEE floating-point spec also requires that the square root function return the best-rounded result. This is interesting because it opens the door to two questions: "How accurately can an irrational function be computed using the IEEE floating-point representation?" and "How should you test the accuracy of a function implementation?"

Computing Accurate Functions

What factors go into determining the accuracy of a computation? The reality is that we should expect to encounter inaccuracies, as round-off can occur in any step in an algorithm. These errors might compound, although sometimes they might also cancel. In practice, an algorithm designer must learn to live with and contain the inherent inaccuracy of numeric computations.

You can separate the factors contributing to computational error into two categories:

1. Factors related to the algorithm used to perform the computation. In particular, this means the stability of the algorithm itself and its sensitivity to round-off errors. A poor, unstable algorithm will tend to return less-accurate results, whereas a good, stable algorithm will return more-accurate results.
2. The intrinsic nature of the function itself and the domain of its inputs. There's a theoretical limit to the achievable accuracy of any function implementation when computation uses a finite (as opposed to an infinite) number of bits. The reason is that round-off acts like a source of error in the computation, and the behavior of the function itself determines whether this error is amplified or attenuated as the calculation proceeds from the inputs to the outputs. For example, computing values of a function near a singularity, a zero or a fast oscillation might be less accurate at those points than computing a function that varies slowly over the same input domain. When talking about this intrinsic attainable accuracy, we speak of how "well conditioned" the function is.

Accordingly, testing the accuracy of a function implementation involves verifying that the algorithm returns results as accurate as theoretically possible. The limits of theoretical possibility are established by the conditioning of the function at each of its inputs. To put it another way, using floating-point numbers to compute a result introduces two possible sources of error: Errors that we can avoid by using good, stable algorithms (factor 1), and errors that are harder to avoid because they're related to the function's behavior at its inputs (factor 2).

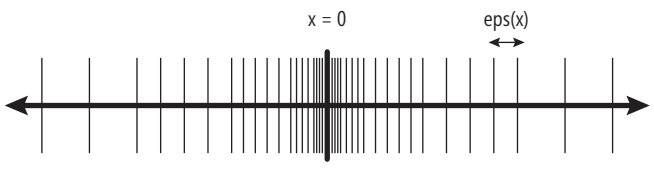


Figure 1 **The Floating-Point Number Grid Depicted Schematically**

In numerical analysis, the concept of conditioning is quantified by the so-called "condition number," which measures the sensitivity of a function to variations in its inputs. Depending on the exact nature of the function under consideration (for example, the number of inputs, scalar versus matrix and so on), there are a number of complicated expressions for the condition number. The simplest case is for a differentiable function of one variable, such as $1/x$, x^2, $sin(x)$ or any of the other functions you likely encountered in high school algebra. In this simple case the condition number of the function f is given by:

$$\kappa_f(x) = \left| \frac{xf'(x)}{f(x)} \right| \qquad \text{(Equation 1)}$$

We'll refer to this later as Equation 1. As always, the notation $f'(x)$ means the derivative of the function $f(x)$. A large condition number ($\kappa_f >> 1$) indicates high sensitivity to relative perturbations, whereas a small condition number ($\kappa_f <= 1$) indicates the function is relatively insensitive to perturbations. Intuitively, you can see that if a function has a singularity—that is, the function blows up, such as $1/x$ near $x = 0$—the derivative will capture this effect and return a large condition number.

For example, consider the function $f(x) = 1/(1-x)$. This function obviously blows up (that is, becomes infinite) at $x = 1$. The derivative is $f'(x) = 1/(1-x)^2$. Plugging this into Equation 1 and eliminating terms gives the condition number for this function:

$$\kappa_f(x) = \frac{x}{1 - x}$$

The condition number $\kappa_f(x)$ goes to infinity around $x = 1$, meaning that computations involving this function will be sensitive to perturbations such as round-off error when x is close to 1. Because $f(x)$ blows up for $x \to 1$, this behavior is expected. Also, the condition number $\kappa_f(x)$ goes to 0 when x is close to 0. This indicates that computations using this function will be insensitive to perturbations when $x \to 0$. This makes intuitive sense because $f(x)$ tends to a constant value of 1 when x is much smaller than 1.

Testing Function Accuracy

So what does a practical test of a mathematical function look like? Suppose we want to test our implementation of the scalar function $y = f(x)$, where x is the input and y is the returned output value. Given a floating-point input value x, testing requires three items:

1. The value computed by the function under test, $y_{computed} = f_{computed}(x)$.
2. The "mathematically true" result. Assume we had an oracle that could tell us the exact, mathematically correct answer. Then round that value to the nearest floating-point value (so it can be represented on a computer). Call this value $y_{true} = f_{true}(x)$.

3. A testing tolerance. For some functions, this tolerance can be 0, which means that $y_{computed}$ is exactly the same as the mathematically true value, y_{true}. For example, the floating-point spec mandates that the function *sqrt()* returns the floating-point value that lies closest to the exact answer (that is, best-rounded). In the general case, however, we can't expect the returns from all functions to be the best-rounded approximations to their exact values. Therefore, the testing tolerance must incorporate information about how much error is allowed in the return from *f(x)*. Note that the tolerance might depend on the details of the function *f(x)*, as well as the exact value of the input, *x*.

With these ingredients, the accuracy of our function implementation is deemed acceptable (that is, it passes our test) if

$$| y_{computed} - y_{true} | < tol(f; x),$$

where the tolerance depends both upon the input value *x* and the behavior of the function *f* itself.

In words, this equation says that the function passes the test if the returned value differs from the "true" value by an amount less than the testing tolerance (the allowed error). Note that $| y_{computed} - y_{true} |$ is the *absolute* error of the computed function.

In this formalism, a function test is performed by creating a large number of input values lying in the input domain of the function, running these values through the function under test and comparing the function outputs $y_{computed}$ against the best-rounded (mathematically true) values, y_{true}. The values of the inputs should be chosen to cover all relevant portions of the function's valid input domain.

At this point, the tester has two questions to answer: What is the "true" result of a function? and What is a reasonable tolerance?

To answer the first question, the easiest thing to do is to use an "infinite precision" math package to create the input/output pairs used for testing. This package doesn't use 32- or 64-bit floating-point numbers to compute a value. Rather, it uses a numeric representation—or better yet, a symbolic representation—of a number that may carry the ability to compute an arbitrarily large number of digits through the computation at the expense of computational speed. Several commercially available mathematics packages implement infinite precision math. Also, many infinite precision math libraries can be plugged into common languages. The ability to use infinite precision math at modern speeds is a recent innovation, making this type of testing convenient. Any of these resources is sufficient to create so-called "golden value" pairs useful for testing a function's floating-point implementation.

Putting It Together—Getting the Testing Tolerance

Once we have golden values, we need to answer the other question: What is a reasonable testing tolerance? Getting the correct testing tolerance is the critical component of testing. If the tolerance is unrealistically small, the function will never pass its test, even if the best algorithm is used to compute it. On the other hand, if the testing tolerance is too large, it means that the allowed error is larger than it needs to be, and the function will pass the test, even if the algorithm is faulty.

The condition number defined earlier is the key to determining the acceptable error allowed in function testing. The expression for the condition number can be rearranged to read:

$$| f(x + \Delta x) - f(x) | \sim \kappa_f(x) \frac{\Delta x}{x} f(x) \qquad \text{(Equation 2)}$$

We'll refer to this later as Equation 2. If we identify Δx as the distance between one floating-point number and the next, this expression tells us how much the output of *f(x)* will jump as we move from one floating-point number *x* to its neighbor, $x + \Delta x$. Traditionally, the field of computer numerics takes for Δx the function *eps(x)*, the spacing between any two adjacent floating-point numbers. Note that because the grid spacing is non-constant (see **Figure 1**), *eps(x)* is a function of *x*. (As noted earlier, the distance between one floating-point number and its nearest neighbor is also related to a ULP—the magnitude represented by the least-significant bit.)

Next, we demand that the output error in our test, $y_{computed} - y_{true}$, be less than some multiple of this jump. That is, we ask that, where *C* is a constant:

$$| y_{computed} - y_{true} | \leq C \kappa_f(x) \frac{eps(x)}{x} f(x)$$

Intuitively, this expression captures the following idea: If *x* makes one step on the floating-point grid, the change in the output should be no greater than Equation 2. An accurate function *f(x)* will change its output by only the amount derived from the condition number, and no more. Therefore, output perturbations occurring during the computation of *f(x)* should be less than the change caused by taking one step on the floating-point grid. The constant *C* is a "fudge factor." Obviously, as *C* increases, the allowed tolerance increases with it. We can therefore interpret this constant as the allowed functional error over the theoretical minimum. In real testing, *C* takes integer values between 1 and 10, which we interpret as the allowed error, expressed in ULPs.

Finally, using the definition of condition number, we can rearrange the equation in a way that exposes the testing tolerance:

$$| y_{computed} - y_{true} | \leq C | f'(x) | eps(x) = \text{tolerance}$$

This is our desired test—it's the expression that must be satisfied by an accurate implementation of the function *f(x)* for any input *x*. In other words, this is the expression used to validate scalar math functions. Note that this expression should hold true for all valid input values *x*.

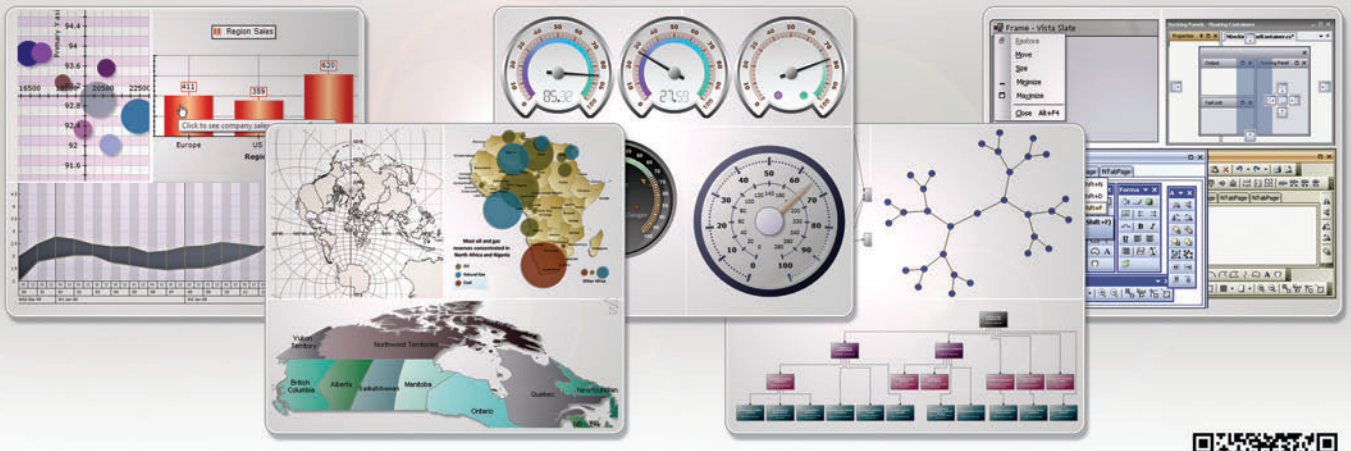
The final question to be settled is, "What is the correct value of *C* to use in testing?" Because we interpret *C* as the number of ULP errors allowed, *C* is an integer of order 1. We set *C* by starting with an initial *C* value (usually 10) and running the test. If the test passes, decrease *C* by 1 and run the test again. Repeat this procedure for decreasing *C* values until the test fails. Then choose the value of *C* that last allowed the test to pass (but always keep a human in the loop to assure reasonableness). The goal of this process is to close the door to passing as much as possible (while still allowing the function to pass the test) so you can be assured the function is tested as rigorously

NEVRON
visualize your success

NEVRON
GAUGE for .NET, SSRS, SharePoint

NEVRON
CHART for .NET, SSRS, SharePoint

NEVRON
USER INTERFACE for .NET



NEVRON
MAP for .NET

NEVRON
DIAGRAM for .NET

2012vol.1 is here

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

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

Download your free evaluation from

www.nevron.com

DEVELOPERS

.NET

IT PROFFESIONALS

SharePoint

SSRS

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

as possible. Well-behaved functions typically require *C* values between 1 and 3, but more complicated functions might require larger *C* values for passing. We find that very few function implementations require *C > 10* to pass, and when we find a function that requires *C > 10*, it often signals a suboptimal implementation.

From the perspective of software testing, once *C* is determined for a function's test (and the test passes), we know that any subsequent test failure means that something in the function implementation has changed—probably for the worse (for example, a regression has occurred). Of course, trade-offs in algorithm design may dictate that it's worth it to give some slack on *C* if the regression is small (for example, a speed versus accuracy trade-off). Likewise, after improving an algorithm, it's worth it to see if *C* can be tightened further. In any event, one job of software testing becomes managing the testing tolerances as the rest of the software is developed.

Golden Value Testing and Beyond

In the testing methodology we've discussed so far, we used precomputed input/output pairs scattered throughout the entire input domain of the function under test. We call this type of testing *golden value testing*, meaning that the infinite precision test oracle provides golden input/output pairs that are known to be highly accurate. The pairs can be stored in a file and read into the test program when it's executed. As a black-box method to test functional accuracy, this approach works quite well. However, other types of function tests also provide important ways to validate function behavior. Other tests include:

- **Special value testing:** Many functions return known, exact theoretical values for certain inputs, such as *cos(0) => 1*, *sin(π) => 0* and *gamma(1/2) = √π*, for examples. In a special value test, known fixed inputs are fed to the function under test, and the function return is compared to the known result. Of course, because irrational numbers such as π don't lie exactly on the floating-point grid, the closest floating-point approximation to those numbers must be used in testing, and a non-zero value for sine and testing tolerance (as computed earlier) is used to validate the computational results.
- **Identity testing:** Many functions obey identities that are true for all inputs over a known domain. For example, *sin(x)^2 + cos(x)^2 == 1* for all inputs *x*. Another one is *arcsin(x) == -i log(i*x + sqrt(1 – x2))*. The difference between an identity test and a special value test is that the identity test is true for arbitrary input, whereas the special value test only holds true for one particular input value. As a consequence, identity tests validate the relationships between functions.

  You must be careful with identity tests. For one thing, some identities are badly conditioned. For example, intermediate results might grow to be quite large even though the final result is moderately sized. In this case, small, allowable errors incurred during intermediate steps in the calculation can cause spurious test failures. Also, many functions (for example, the inverse trig functions) are multivalued in the complex plane. Blindly using identity tests might cause spurious test failures when the left- and right-hand sides of the identity return correct values lying on different Riemann sheets in the complex plane. The tester must carefully craft identity tests to avoid both of these problems.
- **Inverse testing:** This involves computing a function composed with its inverse and verifying the result is the same as the input to the test. For example, for positive *x* we can test *log(exp(x)) = x*. You might think of this as a special case of identity testing, and indeed it is. However, because so many functions have inverses—and mathematical consistency demands that a function composed with its inverse return the original input—we use a distinct test category to validate that functions and their inverses behave in consistent ways. The same caveats that apply to identity tests (such as conditioning or returns lying on different Riemann sheets) also apply to inverse testing.
- **Series summation testing:** Almost all transcendental functions admit a series expansion over some input domain. In series summation testing, we compare the value returned by the function under test to the value returned by an explicitly summed series in the test. You might regard series summation as a type of identity test. However, the considerations going into creating a good series summation test are similar across functions (for example, stability and convergence criteria). Therefore, we consider this type of testing conceptually different from identity testing.

> The ability to use infinite precision math at modern speeds is a recent innovation.

- **Continued fraction expansions:** Some functions may also be evaluated over a known domain using a continued fraction expansion. When such an expansion exists, it might converge much more quickly than a series sum. Therefore, for certain functions, the series summation test might be substituted with a continued fraction expansion.
- **Constructed value testing (also known as model-based testing):** In constructed value testing, you create a simple, verifiably correct implementation of the function within the test itself. This type of test doesn't necessarily apply to scalar math functions. However, a good math package isn't limited to the set of analytic functions. For example, consider functions such as floor, ceiling, mod, concatenation operators for vectors and so on. Each of these functions may be tested by writing code that implements the function using basic computer language constructs. For example, floor, ceiling and the various numeric chop functions may be modeled using cast operators that convert the floating-point value to an integer and then convert the result back to a float (with some simple math performed to mimic the exact behavior

Visual Studio LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

LIVE! 360
IT EVENTS WITH PERSPECTIVE

Orlando, FL **December 10-14**

Royal Pacific Resort at Universal Orlando | **vslive.com/orlando**

Visual Studio Live! provides education and training on what's now, new and next on the .NET development platform.

Session Topics Include:

- HTML5
- Windows 8 / WinRT
- Mobile
- WPF / Silverlight
- Visual Studio / .NET
- XAML

Save Up To $400!

Register Before November 7

Use Promo Code VSOCT

vslive.com/orlando

LIVE! 360
IT EVENTS WITH PERSPECTIVE

Buy 1 Event, Get 3 Free!

Visual Studio Live! is co-located with:

SharePoint LIVE! TRAINING FOR COLLABORATION

SQL Server LIVE! TRAINING FOR DBAs AND IT PROS

Cloud & Virtualization LIVE! THE FUTURE OF COMPUTING

GOLD SPONSOR axosoft

SUPPORTED BY Microsoft msdn Visual Studio MAGAZINE Redmond MAGAZINE

MEDIA SPONSOR THE CODE PROJECT WWW.CODEPROJECT.COM

PRODUCED BY 1105 MEDIA

Figure 2 **Passing ULP Values**

Function	Input Domain	Passing ULP	Comment
Atan2	Real numbers	1	Two input, four quadrant arctan
Cot	Real numbers	3	
Coth	Complex numbers	3	
Cuberoot	Real numbers	2	
Ellipe	Real numbers x such that 0 ≤ x ≤ 1	4	Elliptic integral E
Ellipk	Real numbers x such that 0 ≤ x ≤ 1	6	Elliptic integral K
Expm1	Real numbers	3	Exp(x) - 1
Gamma	Real numbers	3	
Log	Real numbers	3	
Log	Complex numbers	3	
Log1p	Real numbers	1	Log(1+x)
Log1p	Complex numbers	1	Log(1+x)
Psi	Real numbers	6	Psi function—derivative of gamma function
Sqrt	Real numbers	0	IEEE spec requires this to be "best rounded"
Tan	Real numbers	1	
Tanh	Real numbers	4	
Tanh	Complex numbers	4	

of the function under consideration). A benefit of this type of test is that the model implementation is usually trivial and is therefore verifiable by inspection.

- **Error testing:** These tests ensure that invalid input is handled properly by returning a clean error that enables a programmer to clearly and quickly understand the problem. For example, when working in real numbers, *sqrt(x)* should return a useful error (or *NaN*) if $x < 0$. Some input errors are known to crash the computer if they aren't handled properly, and the tester's job is to make sure this doesn't occur.

The benefit of these types of tests is that they provide a cross-check of the function's correctness using only mathematics itself. Because these methods use the self-consistency of mathematics as the foundation of testing, they're "known good" and don't rely on numeric computations using third-party software.

Validating the Approach

One way to appreciate the advantage of condition number tolerances is to examine the tolerances used to validate scalar math functions that have singularities in their input domains. For example, the common trig functions tan and cot are singular at an infinite number of points along the real number line. If fixed, constant tolerances were used for accuracy validation, either these functions would fail their tests for input values close to the singularities, or the testing tolerances necessarily would be so large that the tests wouldn't be effective. With condition number tolerances, we can validate the scalar functions to within a small handful of ULPs over the entire input domain of the function. Shown in **Figure 2** are examples of passing ULP values we've found during qualification of functions distributed in the Microsoft Cloud Numerics product. That is, these are the ULP values at which the corresponding function passes its test.

The conclusion to be drawn from the examples in **Figure 2** is that the special functions are implemented using algorithms that are accurate to within a small number of ULPs. The worst cases presented in **Figure 2** have an error less than 6 ULPs, which corresponds to *log10(2^6) = 1.8* decimal digits of error in the lowest-order digits of the number. For a floating-point double, this corresponds to a relative error of 1.3323e-015.

State-of-the-Art Testing

To review, a rigorous testing methodology has been successfully applied to validation of the algorithms used in the Microsoft Cloud Numerics math packages. Using testing tolerances derived from the function's condition number has enabled us to identify and correct incorrect function implementations, even when the errors occurred only in the last few digits of the function returns. Fixed testing tolerances can't provide a stringent test that doesn't also cause spurious failures (such as near function singularities). The majority of our testing involved using golden value input/output values that are precomputed using infinite precision math packages and then stored in files that are read at test time. In addition to using golden values, we also performed cross-checks on our function implementations using a variety of different math identities to validate the behavior of our functions.

We believe users of Microsoft Cloud Numerics can use its array of math and statistics library functions and be confident that the functions have been thoroughly vetted using state-of-the-art software testing practices.

For more information, we encourage you to read David Goldberg's classic description of floating-point math, "What Every Computer Scientist Should Know About Floating-Point Arithmetic," which is available at bit.ly/vBhP9m and other sites. ■

Stuart Brorson *is an SDET at the Microsoft NERD Center in Cambridge, Mass. He joined Microsoft when the company acquired Interactive Supercomputing, which developed software allowing users to run mathematical and matrix computations on parallel supercomputers. For fun, Brorson enjoys hacking electronics and playing Irish fiddle around Boston.*

Ben Moskowitz *is an SDET specializing in testing mathematics-based projects at Microsoft. He has contributed to the releases of Cloud Numerics and Solver Foundation, an optimization package and platform that won the Visual Studio Lighthouse Award in 2011. Off-hours, he spends time with his wife, caring for their city-dwelling goats.*

Alan Edelman *is a professor of mathematics and a member of the Computer Science and AI Laboratories at MIT in Cambridge, Mass. Professor Edelman has won numerous prizes for his work on numerical analysis, parallel computing and random matrix theory. He was the founder of Interactive Supercomputing, is one of the founders of the Julia project, has learned much from Velvel Kahan and has participated heavily in the Numerical Mathematics Consortium on issues of floating-point mathematics.*

Thanks to the following technical expert for reviewing this article: *Paul Ringseth*

SharePoint LIVE!

TRAINING FOR COLLABORATION

LIVE! 360

IT EVENTS WITH PERSPECTIVE

Orlando, FL December 10-14

Royal Pacific Resort at Universal Orlando | **splive360.com**

SharePoint Live! provides training for those who must customize, deploy and maintain SharePoint Server and SharePoint Foundation to maximize the business value. Both current & newly released versions will be covered!

Session Topics Include:

- Strategy, Governance, Adoption
- Management and Administration
- Information & Content Management
- BI, BPA, Search, and Social
- SharePoint and the Cloud
- SharePoint 2010 & SharePoint 2013

Save Up To $400!

Register Before November 7

Use Promo Code SPOCT

splive360.com

LIVE! 360

IT EVENTS WITH PERSPECTIVE

Buy 1 Event, Get 3 Free!

SharePoint Live! is co-located with:

Cloud & Virtualization LIVE! — THE FUTURE OF COMPUTING

SQL Server LIVE! — TRAINING FOR DBAs AND IT PROS

Visual Studio LIVE! — EXPERT SOLUTIONS FOR .NET DEVELOPERS

GOLD SPONSOR axosoft

SUPPORTED BY Microsoft msdn VIRTUALIZATION REVIEW Redmond MAGAZINE

MEDIA SPONSOR THE CODE PROJECT WWW.CODEPROJECT.COM

PRODUCED BY 1105 MEDIA

JAMES MCCAFFREY

Neural Network Back-Propagation for Programmers

An artificial neural network can be thought of as a meta-function that accepts a fixed number of numeric inputs and produces a fixed number of numeric outputs. In most situations, a neural network has a layer of hidden neurons where each hidden neuron is fully connected to the input neurons and the output neurons. Associated with each individual hidden neuron and each individual output neuron are a set of weight values and a single so-called bias value. The weights and biases determine the output values for a given set of input values.

When neural networks are used to model a set of existing data so that predictions can be made on new data, the main challenge is to find the set of weight and bias values that generate the outputs that best match the existing data. The most common technique for estimating optimal neural network weights and biases is called back-propagation. Although there are many excellent references that describe the complicated mathematics that underlie back-propagation, there are very few guides available for programmers that clearly explain how to program the back-propagation algorithm. This article explains how to implement back-propagation. I use the C# language, but you should have no trouble refactoring the code presented here to other languages.

The best way to see where I'm headed is to take a look at the screenshot of a demo program in **Figure 1**. The demo program creates a neural network that has three input neurons, a hidden layer with four neurons, and two output neurons. Neural networks with a single hidden layer need two activation functions. In many situations, though, the two activation functions are the same, typically the sigmoid function. But in this demo, in order to illustrate the relationship between activation functions and back-propagation, I use different activation functions: the sigmoid function for the input-to-hidden computations, and the tanh (hyperbolic tangent) function for the hidden-to-output computations.

A fully connected 3-4-2 neural network requires 3*4 + 4*2 = 20 weight values and 4+2 = 6 bias values for a total of 26 weights and biases. These weights and biases are initialized to more or less arbitrary values. The three dummy input values are set to 1.0, 2.0 and 3.0. With the initial weight, bias and input values, the initial output values are computed by the neural network to be {0.7225, -0.8779}. The demo program arbitrarily assumes that the two correct output values are {-0.8500, 0.7500}. The goal of the back-propagation algorithm is to find a new set of weights and biases that generate outputs that are very close to the correct values for inputs {1.0, 2.0, 3.0}.

Back-propagation requires two free parameters. The learning rate, usually given the Greek letter eta in back-propagation literature, controls how fast the algorithm converges to a final estimate. The momentum, usually given the Greek letter alpha, helps the back-propagation algorithm avoid situations in which the algorithm oscillates and never converges to a final estimate. The demo program sets the learning rate to 0.90 and the momentum to 0.04. Typically these values are found by trial and error.

> There are very few guides available for programmers that clearly explain how to program the back-propagation algorithm.

Finding the best set of weights and biases for a neural network is sometimes called training the network. Training with back-propagation is an iterative process. At each iteration, back-propagation computes a new set of neural network weight and bias values that in theory generate output values that are closer to the target values. After the first training iteration of the demo program, the back-propagation algorithm found new weight and bias values that generated new outputs of {-0.8932, -0.8006}. The new first output value of -0.8932 was much closer to the first target output value of -0.8500. The second new output value of -0.8006 was still far away from its target value of 0.7500.

The training process can be terminated in a variety of ways. The demo program iterates training until the sum of the absolute differences between output values and target values is <= 0.01 or the training reaches 1,000 iterations. In the demo, after six iterations of training, back-propagation found a set of neural network weight and bias values that generated outputs of {-0.8423, 0.7481}, which were very close to the {-0.8500, 0.7500} desired target values.

This article assumes you have expert-level programming skills and that you have a very basic understanding of neural networks. (For basic information on neural networks, see my May 2012 article, "Dive into Neural Networks," at msdn.microsoft.com/magazine/hh975375.)

This article uses the Visual Studio 2012 beta.

Code download available at archive.msdn.microsoft.com/mag201210TestRun.

SQL Server® LIVE!

TRAINING FOR DBAs AND IT PROS

LIVE! 360
IT EVENTS WITH PERSPECTIVE

Orlando, FL December 10-14

Royal Pacific Resort at Universal Orlando | **sqllive360.com**

SQL Server Live! provides comprehensive education on SQL Server database management, data warehouse/BI model design, Big Data, analytics, performance tuning, troubleshooting and coding against SQL Server.

Session Topics Include:

- BI and Big Data
- Hadoop
- T-SQL, SSIS Packages, Disaster Recovery
- Monitoring, Maintaining, Tuning
- SQL Azure & Windows Azure
- SQL Server 2012

Save Up To $400!

Register Before November 7

Use Promo Code SQOCT

sqllive360.com

LIVE! 360
IT EVENTS WITH PERSPECTIVE

Buy 1 Event, Get 3 Free!

SQL Server Live! is co-located with:

SharePoint LIVE! TRAINING FOR COLLABORATION

Cloud & Virtualization LIVE! THE FUTURE OF COMPUTING

Visual Studio LIVE! EXPERT SOLUTIONS FOR .NET DEVELOPERS

GOLD SPONSOR axosoft

SUPPORTED BY Microsoft msdn VIRTUALIZATION REVIEW Redmond MAGAZINE

MEDIA SPONSOR THE CODE PROJECT WWW.CODEPROJECT.COM

PRODUCED BY 1105 MEDIA

The code for the demo program shown in **Figure 1** is a bit too long to present in this article, so I'll concentrate on explaining the key parts of the algorithm. The complete source code for the demo program is available at archive.msdn.microsoft.com/mag201210TestRun.

Defining a Neural Network Class

Coding a neural network that uses back-propagation lends itself nicely to an object-oriented approach. The class definition used for the demo program is listed in **Figure 2**.

Member fields, numInput, numHidden and numOutput are defining characteristics of the neural network architecture. In addition to a simple constructor, the class has four publicly accessible methods and two helper methods. Method UpdateWeights contains all the logic of the back-propagation algorithm. Method SetWeights accepts an array of weights and biases and copies those values sequentially into member arrays. Method GetWeights performs the reverse operation by copying the weights and biases into a single array and returning that array. Method ComputeOutputs determines the neural network output values using the current input, weight and bias values.

Method SigmoidFunction is used as the input-to-hidden activation function. It accepts a real value (type double in C#) and returns a value between 0.0 and 1.0. Method HyperTanFunction also accepts a real value but returns a value between -1.0 and +1.0. The C# language has a built-in hyperbolic tangent function, Math.Tanh, but if you're using a language that doesn't have a native tanh function, you'll have to code one from scratch.

Setting up the Arrays

One of the keys to successfully programming a neural network back-propagation algorithm is to fully understand the arrays that are being used to store weight and bias values, store different kinds of input and output values, store values from a previous iteration of the algorithm, and store scratch calculations. The large diagram in **Figure 3** contains all the information you need to know to understand how to program back-propagation. Your initial reaction to **Figure 3** is likely to be something along the lines of, "Forget it—this is too complicated." Hang in there. Back-propagation is not trivial, but once you understand the diagram you'll be able to implement back-propagation using any programming language.

Figure 3 has primary inputs and outputs at the edges of the figure, but also several local input and output values that occur in the interior of the diagram. You should not underestimate the difficulty of coding a neural network and the need to keep the names and meanings of

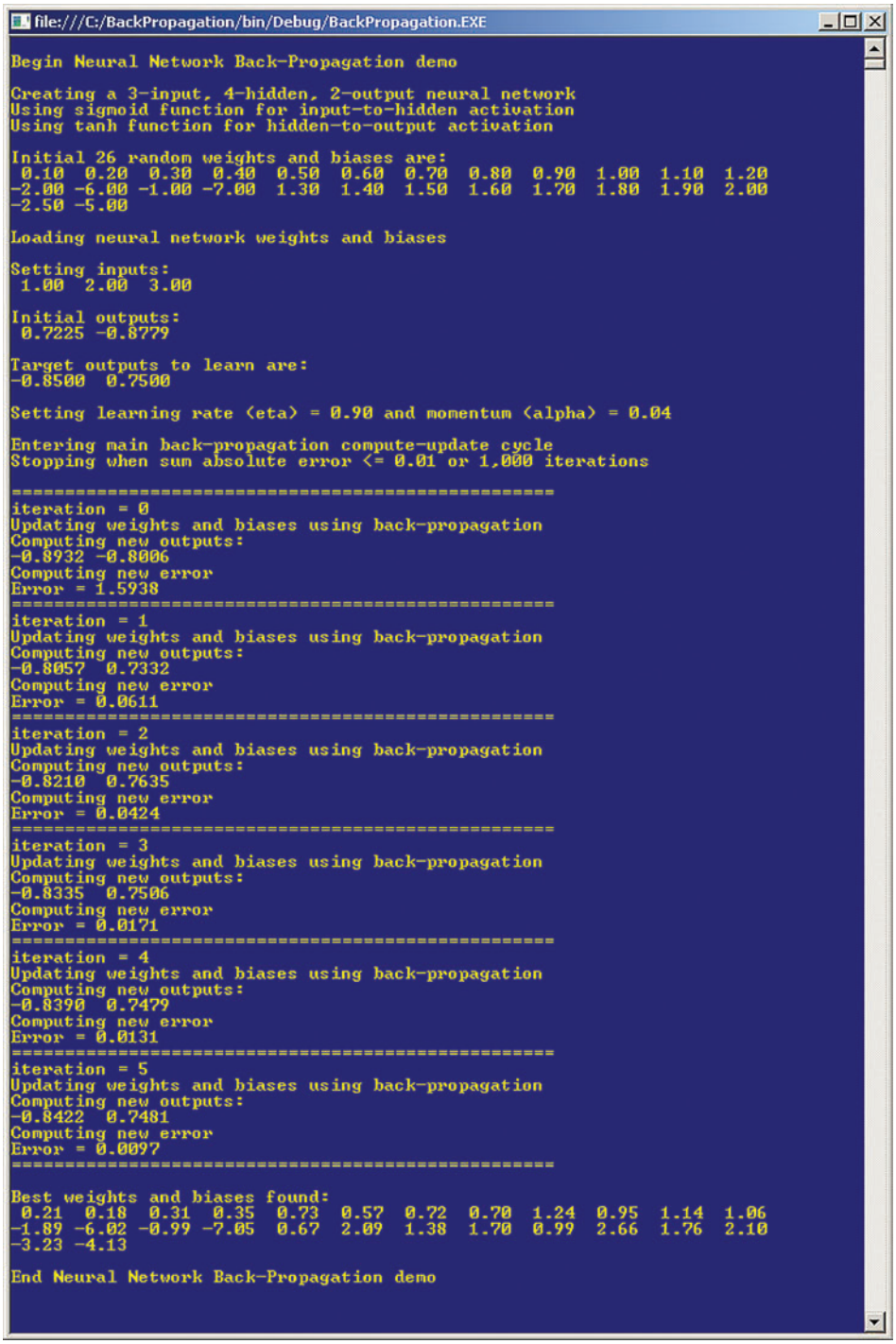


Figure 1 **Back-Propagation Algorithm in Action**

Cloud & Virtualization LIVE!

THE FUTURE OF COMPUTING

LIVE! 360
IT EVENTS WITH PERSPECTIVE

Orlando, FL **December 10-14**

Royal Pacific Resort at Universal Orlando | **virtlive360.com**

Cloud & Virtualization Live! is a new event that provides in-depth training with real-world applicability on today's cloud and virtualization technologies.

Session Topics Include:

- Becoming a Virtualization Expert
- Managing the Modern Cloud Enabled Datacenter
- Tactics in Cloud Application Development
- Constructing & Managing an Application Delivery Infrastructure
- Integrating Mobile Devices and Consumerization with Cloud Computing

Save Up To $400!

Register Before November 7

Use Promo Code CVOCT

virtlive360.com

LIVE! 360
IT EVENTS WITH PERSPECTIVE

Buy 1 Event, Get 3 Free!

Cloud & Virtualization Live! is co-located with:

SharePoint LIVE! TRAINING FOR COLLABORATION

SQL Server LIVE! TRAINING FOR DBAs AND IT PROS

Visual Studio LIVE! EXPERT SOLUTIONS FOR .NET DEVELOPERS

GOLD SPONSOR axosoft

SUPPORTED BY Microsoft msdn VIRTUALIZATION REVIEW Redmond MAGAZINE

MEDIA SPONSOR THE CODE PROJECT WWW.CODEPROJECT.COM

PRODUCED BY 1105 MEDIA

all these inputs and outputs clear. Based on my experience, a diagram like the one in **Figure 3** is absolutely essential.

The first five of the 15 arrays used in the neural network definition outlined in **Figure 2** deal with the input-to-hidden layers and are:

```
public class NeuralNetwork
{
  // Declare numInput, numHidden, numOutput
  private double[] inputs;
  private double[][] ihWeights;
  private double[] ihSums;
  private double[] ihBiases;
  private double[] ihOutputs;
...
```

The first array, named inputs, holds the numeric input values. These values typically come directly from some normalized data source such as a text file. The NeuralNetwork constructor instantiates inputs as:

```
this.inputs = new double[numInput];
```

Array ihWeights (input-to-hidden weights) is a virtual two-dimensional array implemented as an array of arrays. The first index indicates the input neuron and the second index indicates the hidden neuron. The array is instantiated by the constructor as:

```
this.ihWeights = Helpers.MakeMatrix(numInput, numHidden);
```

Here, Helpers is a utility class of static methods that help simplify the neural network class:

```
public static double[][] MakeMatrix(int rows, int cols)
{
  double[][] result = new double[rows][];
  for (int i = 0; i < rows; ++i)
    result[i] = new double[cols];
  return result;
}
```

Array ihSums is a scratch array that's used to hold an intermediate calculation in the ComputeOutputs method. The array holds values that will become the local inputs for the hidden neurons and is instantiated as:

```
this.ihSums = new double[numHidden];
```

Array ihBiases holds the bias values for the hidden neurons. Neural network weight values are constants that are applied by multiplying them with a local input value. Bias values are added to an intermediate sum to produce a local output value, which becomes the local input to the next layer. Array ihBiases is instantiated as:

```
this.ihBiases = new double[numHidden];
```

Array ihOutputs holds the values that are emitted from the hidden-layer neurons (which become the inputs to the output layer).

The next four arrays in the NeuralNetwork class hold values related to the hidden-to-output layer:

```
private double[][] hoWeights;
private double[] hoSums;
private double[] hoBiases;
private double[] outputs;
```

These four arrays are instantiated in the constructor as:

```
this.hoWeights = Helpers.MakeMatrix(numHidden, numOutput);
this.hoSums = new double[numOutput];
this.hoBiases = new double[numOutput];
this.outputs = new double[numOutput];
```

> Finding the best set of weights and biases for a neural network is sometimes called training the network.

The neural network class has six arrays that are directly related to the back-propagation algorithm. The first two arrays hold values called the gradients for the output- and hidden-layer neurons. A gradient is a value that indirectly describes how far off, and in what direction (positive or negative), local outputs are relative to the target outputs. Gradient values are used to compute delta values, which are added to current weight and bias values to produce new, better weights and biases. There's one gradient value for each hidden-layer neuron and each output-layer neuron. The arrays are declared as:

```
private double[] oGrads; // Output gradients
private double[] hGrads; // Hidden gradients
```

The arrays are instantiated in the constructor as:

```
this.oGrads = new double[numOutput];
this.hGrads = new double[numHidden];
```

The final four arrays in class NeuralNetwork hold the deltas (not gradients) from the previous iteration of the training loop. These previous deltas are required if you use the momentum mechanism to prevent back-propagation non-convergence. I consider momentum essential, but if you decide not to implement momentum you can omit these arrays. They are declared as:

```
private double[][] ihPrevWeightsDelta;  // For momentum
private double[] ihPrevBiasesDelta;
private double[][] hoPrevWeightsDelta;
private double[] hoPrevBiasesDelta;
```

These arrays are instantiated as:

```
ihPrevWeightsDelta = Helpers.MakeMatrix(numInput, numHidden);
ihPrevBiasesDelta = new double[numHidden];
hoPrevWeightsDelta = Helpers.MakeMatrix(numHidden, numOutput);
hoPrevBiasesDelta = new double[numOutput];
```

Computing Outputs

Each iteration in the training loop shown in **Figure 1** has two parts. In the first part, outputs are computed using the current primary

Figure 2 **Neural Network Class**

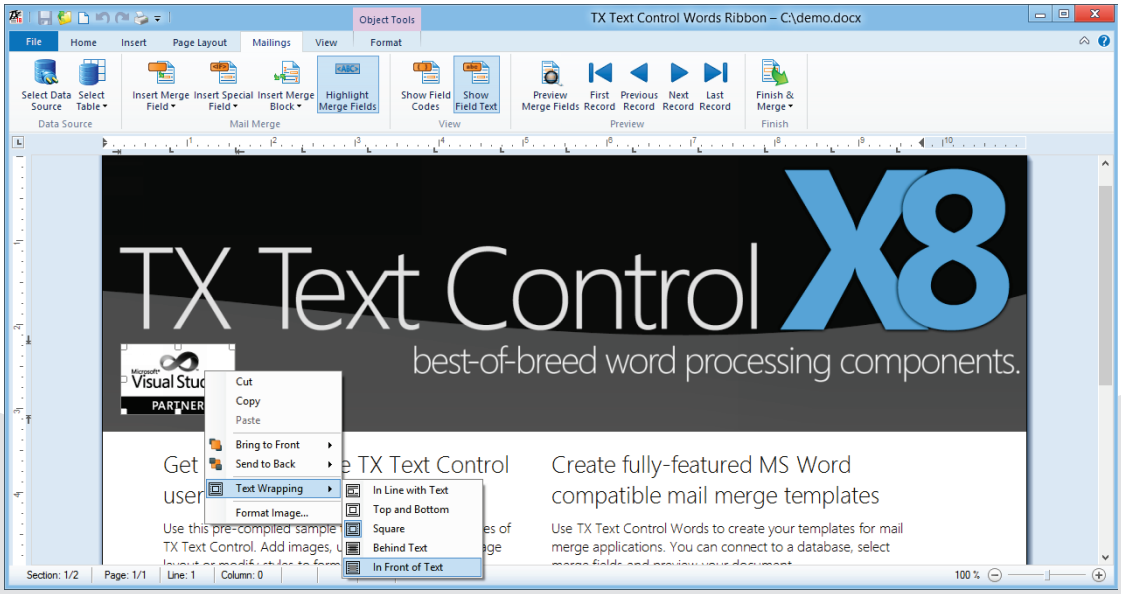
```
class NeuralNetwork
{
  private int numInput;
  private int numHidden;
  private int numOutput;

  // 15 input, output, weight, bias, and other arrays here

  public NeuralNetwork(int numInput, int numHidden, int numOutput) {...}
  public void UpdateWeights(double[] tValues, double eta, double alpha) {...}
  public void SetWeights(double[] weights) {...}
  public double[] GetWeights() {...}
  public double[] ComputeOutputs(double[] xValues) {...}

  private static double SigmoidFunction(double x)
  {
    if (x < -45.0) return 0.0;
    else if (x > 45.0) return 1.0;
    else return 1.0 / (1.0 + Math.Exp(-x));
  }

  private static double HyperTanFunction(double x)
  {
    if (x < -10.0) return -1.0;
    else if (x > 10.0) return 1.0;
    else return Math.Tanh(x);
  }
}
```

WINDOWS FORMS | WPF | ASP.NET | ACTIVEX

WORD PROCESSING COMPONENTS

Ⓘ Rich Text Editing

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



PDF Reflow - Open PDF Documents

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

Free License

Download our 100% free Express version.

Spell Checking

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

Reporting Redefined

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

TX TEXT CONTROL® word processing components

Microsoft® Visual Studio® CONNECTIONS
OCT 29 - Nov 1, 2012, LAS VEGAS
Booth #303

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

WWW.TEXTCONTROL.COM

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

inputs, weights and biases. In the second part, back-propagation is used to modify the weights and biases. The diagram in **Figure 3** illustrates both parts of the training process.

Working from left to right, inputs x0, x1 and x2 are assigned values of 1.0, 2.0 and 3.0. These primary input values go into the input-layer neurons and are emitted without modification. Although input-layer neurons can modify their input, such as normalizing the values to be within a certain range, in most cases such processing is done externally. Because of this, neural network diagrams often use rectangles or square boxes for the input neurons to indicate they aren't processing neurons in the same sense that the hidden-layer and output-layer neurons are. Additionally, this affects the terminology used. In some cases, the neural network shown in **Figure 3** would be called a three-layer network, but because the input layer doesn't perform processing, the neural network shown is sometimes called a two-layer network.

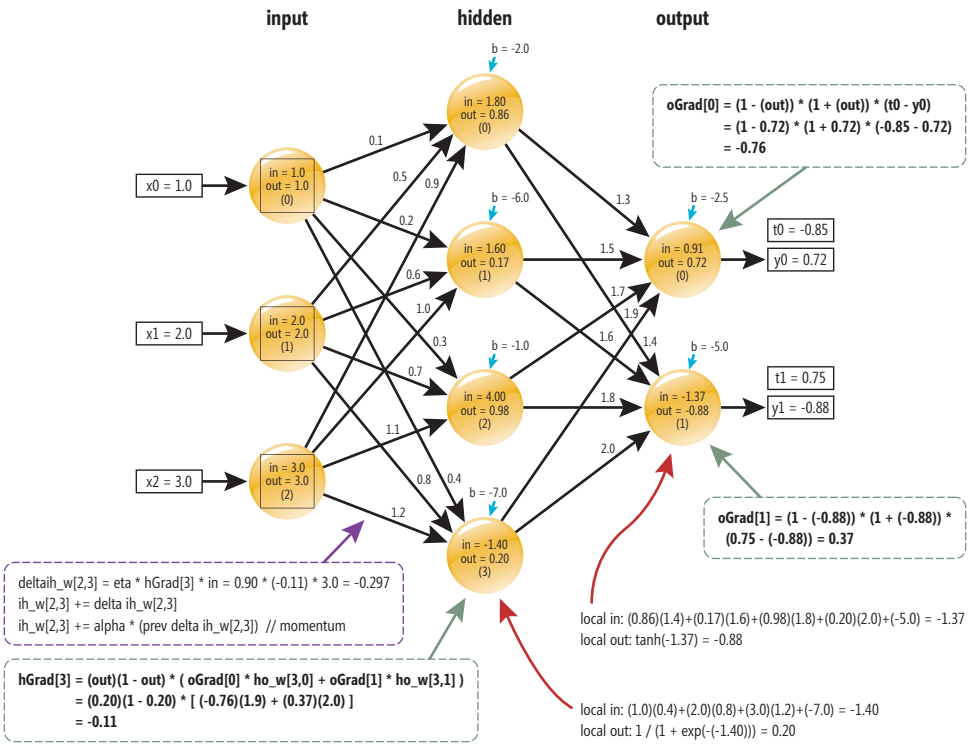


Figure 3 **The Back-Propagation Algorithm**

Next, each of the hidden-layer neurons computes a local input and a local output. For example, the bottommost hidden neuron, with index [3], computes its scratch sum as (1.0)(0.4)+(2.0)(0.8)+(3.0)(1.2) = 5.6. The scratch sum is the product of the sum of the three inputs times the associated input-to-hidden weight. The values above each arrow are the weights. Next, the bias value, -7.0, is added to the scratch sum to yield a local input value of 5.6 + (-7.0) = -1.40. Then the input-to-hidden activation function is applied to this intermediate input value to yield the local output value of the neuron. In this case, the activation function is the sigmoid function, so the local output is 1 / (1 + exp(-(-1.40))) = 0.20.

The output-layer neurons compute their input and output similarly. For example, in **Figure 3**, the bottommost output-layer neuron with index[1] computes its scratch sum as (0.86)(1.4)+(0.17)(1.6)+(0.98)(1.8)+(0.20)(2.0) = 3.73. The associated bias is added to give the local input: 3.73 + (-5.0) = -1.37. And the activation function is applied to give the primary output: tanh(-1.37) = -0.88. If you examine the code for ComputeOutputs, you'll see that the method computes outputs exactly as I've just described.

Back-Propagation

Although the math behind the theory of back-propagation is fairly complicated, once you know what those math results are, implementing back-propagation is not too difficult. Back-propagation starts by working from right to left in the diagram shown in **Figure 3**. The first step is to compute the gradient values for each output-layer neuron. Recall the gradient is a value that has information regarding the magnitude and direction of an error. The gradients for the output-layer neurons are computed differently from the gradients for the hidden-layer neurons.

The gradient of an output-layer neuron is equal to the target (desired) value minus the computed output value, times the calculus derivative of the output-layer activation function evaluated at the computed output value. For example, the gradient value of the bottommost output-layer neuron in **Figure 3**, with index [1], is computed as:

```
(0.75 - (-0.88)) * (1 - (-0.88)) * (1 + (-0.88)) = 0.37
```

> Compared to alternatives such as particle swarm optimization and evolutionary optimization algorithms, back-propagation tends to be faster.

The 0.75 is the desired value. The -0.88 is the computed output value from the forward-pass computation. Recall that in this example the output-layer activation function is the tanh function. The calculus derivative of tanh(x) is (1 - tanh(x)) * (1 + tanh(x)). The math analysis is a bit tricky but, ultimately, computing the gradient of an output-layer neuron is given by the formula described here.

The gradient of a hidden-layer neuron is equal to the calculus derivative of the activation function of the hidden layer evaluated

at the local output of the neuron times the sum of the product of the primary outputs times their associated hidden-to-output weights. For example, in **Figure 3**, the gradient of the bottommost hidden-layer neuron with index [3] is:

```
(0.20)(1 - 0.20) * [ (-0.76)(1.9) + (0.37)(2.0) ] = -0.03
```

If we call the sigmoid function g(x), it turns out that the calculus derivative of the sigmoid function is g(x) * (1 - g(x)). Recall that this example uses the sigmoid function for the input-to-hidden activation function. Here the 0.20 is the local output from the neuron. The -0.76 and 0.37 are the gradients of the output-layer neurons, and the 1.9 and 2.0 are the hidden-to-output weights associated with the two output-layer gradients.

Computing the Weight and Bias Deltas

After all the output-layer gradients and hidden-layer gradients have been computed, the next step in the back-propagation algorithm is to use the gradient values to compute delta values for each weight and bias value. Unlike the gradients, which must be computed right to left, the delta values can be computed in any order. The delta value for any weight or bias is equal to eta times the gradient associated with the weight or bias, times the input value associated with the weight or bias. For example, the delta value for the input-to-hidden weight from input neuron [2] to hidden neuron [3] is:

```
delta i-h weight[2][3] = eta * hidden gradient[3] * input[2]
= 0.90 * (-0.11) * 3.0
= -0.297
```

The 0.90 is eta, which controls how fast the back-propagation learns. Larger values of eta produce larger changes in delta, with the risk of overshooting a good answer. The -0.11 value is the gradient for hidden neuron [3]. The 3.0 value is the input value for input neuron [2]. In terms of the diagram in **Figure 3**, if a weight is represented as an arrow from one neuron to another, to compute the delta for a particular weight, you use the gradient value of the neuron pointed to on the right and the input value of the neuron pointed from on the left.

When computing the deltas for bias values, notice that because bias values are simply added to an intermediate sum, they have no associated input value. So, to compute the delta for a bias value you can either omit the input value term altogether, or use a dummy 1.0 value as a form of documentation. For example, in **Figure 3**, the bottommost hidden-layer bias has value -7.0. The delta for that bias value is:

```
0.90 * gradient of neuron pointed to * 1.0
= 0.90 * (-0.11) * 1.0
= 0.099
```

Adding a Momentum Term

After all weight and bias delta values have been computed, it's possible to update each weight and bias by simply adding the associated delta value. However, experience with neural networks has shown that with certain data sets, the back-propagation algorithm can oscillate, repeatedly overshooting and then undershooting the target value and never converging to a final set of weight and bias estimates. One technique for reducing this tendency is to add to each new weight and bias an additional term called momentum. The momentum for a weight (or bias) is just some small value (like 0.4 in the demo program) times the value of the previous delta for the weight. Using momentum adds a small amount of complexity to the back-propagation algorithm because the values of previous deltas must be stored. The math behind why this technique prevents oscillation is subtle, but the result is simple.

To summarize, to update a weight (or bias) using back-propagation, the first step is to compute gradients for all output-layer neurons. The second step is to compute gradients for all hidden-layer neurons. The third step is to compute deltas for all weights using eta, the learning rate. The fourth step is to add the deltas to each weight. The fifth step is to add a momentum term to each weight.

Coding with Visual Studio 2012

The explanation of back-propagation presented in this article, together with the sample code, should give you enough information to understand and code the back-propagation algorithm. Back-propagation is just one of several techniques that can be used to estimate the best weight and bias values for a data set. Compared to alternatives such as particle swarm optimization and evolutionary optimization algorithms, back-propagation tends to be faster. But back-propagation does have disadvantages. It can't be used with neural networks that use non-differentiable activation functions. Determining good values for the learning rate and momentum parameters is more art than science and can be time-consuming.

> I was pleasantly surprised that I wasn't unpleasantly surprised by any of the new features in Visual Studio 2012.

There are several important topics that this article does not address, in particular how to deal with multiple target data items. I'll explain this concept and other neural network techniques in future articles.

When I coded the demo program for this article, I used the beta version of Visual Studio 2012. Even though many of the new features in Visual Studio 2012 are related to Windows 8 apps, I wanted to see how Visual Studio 2012 handled good old console applications. I was pleasantly surprised that I wasn't unpleasantly surprised by any of the new features in Visual Studio 2012. My transition to Visual Studio 2012 was effortless. Although I didn't make use of the new Async feature in Visual Studio 2012, it could have been useful when computing all the delta values for each weight and bias. I tried out the new Call Hierarchy feature and found it useful and intuitive. My initial impressions of Visual Studio 2012 were favorable, and I plan to transition to it as soon as I'm able. ■

DR. JAMES MCCAFFREY *works for Volt Information Sciences Inc., where he manages technical training for software engineers working at the Microsoft Redmond, Wash., campus. He has worked on several Microsoft products including Internet Explorer and MSN Search. He's the author of ".NET Test Automation Recipes" (Apress, 2006), and can be reached at jammc@microsoft.com.*

THANKS to the following technical expert for reviewing this article: *Dan Liebling*

TED NEWARD

Cassandra NoSQL Database, Part 2: Programming

In my August 2012 column, "Cassandra NoSQL Database: Getting Started," I examined Apache Cassandra. It's described as the "open source, distributed, decentralized, elastically scalable, highly available, fault-tolerant, tuneably consistent, column-oriented database that bases its distribution design on Amazon Dynamo and its data model on Google Bigtable" in the book, "Cassandra: The Definitive Guide" (O'Reilly Media, 2010). To be more precise, I looked at how to install Cassandra (which, because it's a Java-based database, also required getting a Java Virtual Machine up and running on your machine if you didn't have one already), how to connect to it from the command line and what its data model looked like. The data model bears repeating because it's quite noticeably different in structure than the relational database with which most developers are familiar.

Cassandra is a "column-oriented" data store.

As I discussed last time (msdn.microsoft.com/magazine/JJ553519), Cassandra is a "column-oriented" data store, which means that instead of storing identically structured tuples of data arranged according to a fixed structure (the table schema), Cassandra stores "column families" in "keyspaces." In more descriptive terms, Cassandra associates a key value with a varying number of name/value pairs (columns) that might be entirely different from one "row" to another.

For example, consider the keyspace "Earth" I created last time, with a column family named "People," into which I'll write rows that (may or may not) look like this:

```
RowKey: tedneward
  ColumnName:"FirstName", ColumnValue:"Ted"
  ColumnName:"LastName", ColumnValue:"Neward"
  ColumnName:"Age", ColumnValue:41
  ColumnName:"Title", ColumnValue:"Architect"
RowKey: rickgaribay
  ColumnName:"FirstName", ColumnValue:"Rick"
  ColumnName:"LastName", ColumnValue:"Garibay"
RowKey: theartistformerlyknownasprince
  ColumnName:"Identifier", ColumnValue: <image>
  ColumnName:"Title", ColumnValue:"Rock Star"
```

As you can see, each "row" contains conceptually similar data, but not all "rows" will have the same data, depending on what the developer or business needed to store for any particular row key. I don't know Rick's age, so I couldn't store it. In a relational database, if the schema mandated that age was a non-NULLABLE column, I couldn't have stored Rick at all. Cassandra says, "Why not?"

Figure 1 **Creating a System Keyspace**

```
[TestMethod]
public void DoesMyKeyspaceExistAndCreateItIfItDoesnt()
{
  using (var db = new CassandraContext(keyspace: "system", server:Server))
  {
    bool foundEarth = false;
    foreach (CassandraKeyspace keyspace in db.DescribeKeyspaces())
    {
      Apache.Cassandra.KsDef def = keyspace.GetDescription();
      if (def.Name == "Earth")
        foundEarth = true;
    }

    if (!foundEarth)
    {
      var keyspace = new CassandraKeyspace(new CassandraKeyspaceSchema
      {
        Name = "Earth"
      }, db);

      keyspace.TryCreateSelf();
    }

    Assert.IsTrue(db.KeyspaceExists("Earth"));
  }
}
```

My previous column demonstrated inserting and removing data from the command line, but this isn't particularly helpful if the goal is to write applications that will access and store data. So, without further background, let's dive into what it takes to write applications that read from and store to Cassandra.

Figure 2 **Creating a Column Family Using Cassandra Query Language**

```
[TestMethod]
public void CreateAColumnFamily()
{
  using (var db = new CassandraContext(keyspace: "Earth", server: Server))
  {
    CassandraColumnFamily cf = db.GetColumnFamily("People");
    if (cf == null)
    {
      db.ExecuteNonQuery(@"CREATE COLUMNFAMILY People (
        KEY ascii PRIMARY KEY,
        FirstName text,
        LastName text,
        Age int,
        Title text
);");
    }

    cf = db.GetColumnFamily("People");
    Assert.IsNotNull(cf);
  }
}
```

Figure 3 **Storing Objects in the Database**

```
[TestMethod]
public void StoreSomeData()
{
  using (var db = new CassandraContext(keyspace: "Earth", server: Server))
  {
    var peopleCF = db.GetColumnFamily("People");
    Assert.IsNotNull(peopleCF);

    Assert.IsNull(db.LastError);

    dynamic tedneward = peopleCF.CreateRecord("TedNeward");
    tedneward.FirstName = "Ted";
    tedneward.LastName = "Neward";
    tedneward.Age = 41;
    tedneward.Title = "Architect";

    db.Attach(tedneward);
    db.SaveChanges();

    Assert.IsNull(db.LastError);
  }
}
```

Cassandra, O Cassandra, Wherefore Art Thou Cassandra?

To start, I need to connect to Cassandra from the Microsoft .NET Framework. Doing so involves one of two techniques: I can use the native Apache Thrift API, or I can use a third-party wrapper on top of the native Thrift API. Thrift is a binary remote procedure call toolkit, similar in many ways to DCOM (bet you haven't thought of that in a few years) or CORBA or .NET Remoting. It's a particularly low-level approach to communicating with Cassandra, and while Thrift has C# support, it's not trivial to get all that up and running. Alternatives to Thrift include FluentCassandra, cassandra-sharp, Cassandraemon and Aquiles (the Spanish translation of Achilles, which keeps the ancient Greek theme alive and well). All of these are open source and offer some nicer abstractions over the Cassandra API. For this column, I'm going to use FluentCassandra, but any of them seem to work pretty well, the odd Internet flame war notwithstanding.

Figure 4 **Storing More People in the Keyspace**

```
[TestMethod]
public void StoreSomeData()
{
  using (var db = new CassandraContext(keyspace: "Earth", server: Server))
  {
    var peopleCF = db.GetColumnFamily("People");
    Assert.IsNotNull(peopleCF);

    Assert.IsNull(db.LastError);

    dynamic tedneward = peopleCF.CreateRecord("TedNeward");
    tedneward.FirstName = "Ted";
    tedneward.LastName = "Neward";
    tedneward.Age = 41;
    tedneward.Title = "Architect";

    dynamic rickgaribay = peopleCF.CreateRecord("RickGaribay");
    rickgaribay.FirstName = "Rick";
    rickgaribay.LastName = "Garibay";
    rickgaribay.HomeTown = "Phoenix";

    dynamic theArtistFormerlyKnownAsPrince =
      peopleCF.CreateRecord("TAFKAP");
    theArtistFormerlyKnownAsPrince.Title = "Rock Star";

    db.Attach(tedneward);
    db.Attach(rickgaribay);
    db.Attach(theArtistFormerlyKnownAsPrince);
    db.SaveChanges();

    Assert.IsNull(db.LastError);
  }
}
```

FluentCassandra is available as a NuGet package, so the easiest way to get started is to fire up the NuGet Package Manager in a Visual Studio Test project (so I can write exploration tests) and do an "Install-Package FluentCassandra." (The most recent version as of this writing is 1.1.0.) Once that's done, and I've double-checked that the Cassandra server is still running after I toyed with it for the August column, I can write the first exploration test: connecting to the server.

FluentCassandra lives in the namespace "FluentCassandra" and two nested namespaces ("Connections" and "Types"), so I'll bring those in, and then write a test to see about connecting to the database:

```
private static readonly Server Server = new Server("localhost");
TestMethod]
public void CanIConnectToCassandra()
{
  using (var db = new CassandraContext(keyspace: "system", server:Server))
  {
    var version = db.DescribeVersion();
    Assert.IsNotNull(version);
    testContextInstance.WriteLine("Version = {0}", version);
    Assert.AreEqual("19.30.0", version);
  }
}
```

Note that by the time you read this, it's possible that the version number will be different from when I wrote it, so if that second assertion fails, check the output window to see the returned string. (Remember, exploration tests are about testing your understanding of the API, so writing output isn't as much of a bad idea as it is in an automated unit test.)

> Each "row" contains conceptually similar data, but not all "rows" will have the same data, depending on what the developer or business needed to store for any particular row key.

The CassandraContext class has five different overloads for connecting to a running Cassandra server, all of them pretty easy to infer—they all deal with connection information of one form or another. In this particular case, because I haven't created the keyspace in which I want to store (and later read) the data, I'm connecting to the "system" keyspace, which is used by Cassandra to store various systemic details in much the same way that most relational databases have one instance reserved for database metadata and security and such. But this means I don't want to write to that system keyspace; I want to create my own, which forms the next exploration test, as shown in **Figure 1**.

Figure 5 **Running a Test**

```
[TestInitialize]
public void Setup()
{
  using (var db = new CassandraContext(keyspace: "Earth", server: Server))
  {
    var keyspace = new CassandraKeyspace(new CassandraKeyspaceSchema {
      Name = "Earth",
      }, db);
    keyspace.TryCreateSelf();

    db.ExecuteNonQuery(@"CREATE COLUMNFAMILY People (
      KEY ascii PRIMARY KEY,
      FirstName text,
      LastName text,
      Age int,
      Title text);");

    var peopleCF = db.GetColumnFamily("People");

    dynamic tedneward = peopleCF.CreateRecord("TedNeward");
    tedneward.FirstName = "Ted";
    tedneward.LastName = "Neward";
    tedneward.Age = 41;
    tedneward.Title = "Architect";

    dynamic rickgaribay = peopleCF.CreateRecord("RickGaribay");
    rickgaribay.FirstName = "Rick";
    rickgaribay.LastName = "Garibay";
    rickgaribay.HomeTown = "Phoenix";

    dynamic theArtistFormerlyKnownAsPrince =
      peopleCF.CreateRecord("TAFKAP");
    theArtistFormerlyKnownAsPrince.Title = "Rock Star";

    db.Attach(tedneward);
    db.Attach(rickgaribay);
    db.Attach(theArtistFormerlyKnownAsPrince);
    db.SaveChanges();
  }
}

[TestCleanup]
public void TearDown()
{
  var db = new CassandraContext(keyspace: "Earth", server: Server);
  if (db.KeyspaceExists("Earth"))
    db.DropKeyspace("Earth");
}
```

Figure 6 **Fetching Data with the Get Method**

```
[TestMethod]
public void StoreAndFetchSomeData()
{
  using (var db = new CassandraContext(keyspace: "Earth", server: Server))
  {
    var peopleCF = db.GetColumnFamily("People");
    Assert.IsNotNull(peopleCF);

    Assert.IsNull(db.LastError);

    dynamic jessicakerr = peopleCF.CreateRecord("JessicaKerr");
    jessicakerr.FirstName = "Jessica";
    jessicakerr.LastName = "Kerr";
    jessicakerr.Gender = "F";

    db.Attach(jessicakerr);
    db.SaveChanges();

    Assert.IsNull(db.LastError);

    dynamic result = peopleCF.Get("JessicaKerr").FirstOrDefault();
    Assert.AreEqual(jessicakerr.FirstName, result.FirstName);
    Assert.AreEqual(jessicakerr.LastName, result.LastName);
    Assert.AreEqual(jessicakerr.Gender, result.Gender);
  }
}
```

Admittedly, the loop through all the keyspaces in the database is unnecessary—I do it here to demonstrate that there are places in the FluentCassandra API where the underlying Thrift-based API peeks through, and the "Apache.Cassandra.KsDef" type is one of those.

Now that I have a keyspace, I need at least one column family within that keyspace. The easiest way to create this uses Cassandra Query Language (CQL), a vaguely SQL-like language, as shown in **Figure 2**.

The danger of CQL is that its deliberately SQL-like grammar combines with the easy misperception that "Cassandra has columns, therefore it must have tables like a relational database" to trick the unwary developer into thinking in relational terms.

The danger of CQL is that its deliberately SQL-like grammar combines with the easy misperception that "Cassandra has columns, therefore it must have tables like a relational database" to trick the unwary developer into thinking in relational terms. This leads to conceptual assumptions that are wildly wrong. Consider, for example, the columns in **Figure 2**. In a relational database, only those five columns would be allowed in this column family. In Cassandra, those are just "guidelines" (in a quaintly "Pirates of the Caribbean" sort of way). But, the alternative (to not use CQL at all) is less attractive by far: Cassandra offers the API TryCreateColumnFamily (not shown), but no matter how many times I try to wrap my head around it, this still feels more clunky and confusing than the CQL approach.

'Data, Data, Data! I Cannot Make Bricks Without Clay!'

Once the column family is in place, the real power of the FluentCassandra API emerges as I store some objects into the database, as shown in **Figure 3**.

Notice the use of the "dynamic" facilities of C# 4.0 to reinforce the idea that the column family is not a strictly typed collection of name/value pairs. This allows the C# code to reflect the nature of the column-oriented data store. I can see this when I store a few more people into the keyspace, as shown in **Figure 4**.

Again, just to drive the point home, notice how Rick has a HomeTown column, which wasn't specified in the earlier description of this column family. This is completely acceptable, and quite common.

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

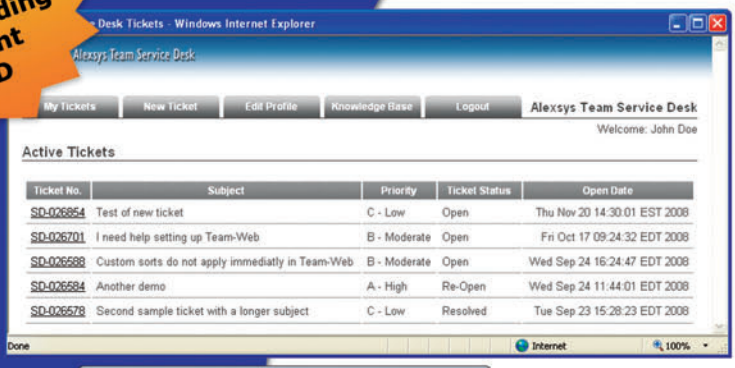
Service Desk Features

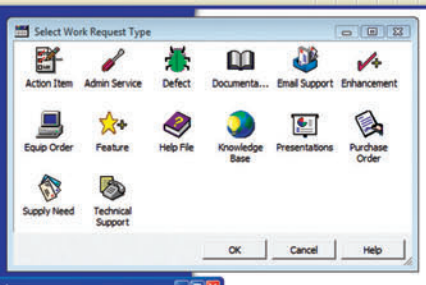
- Fully Secure
- Unlimited Users Self Registered or Active Directory
- Integrated into Your Web Site
- Fast/AJAX Dynamic Content
- Unlimited Service Desks
- Visual Service Desk Builder

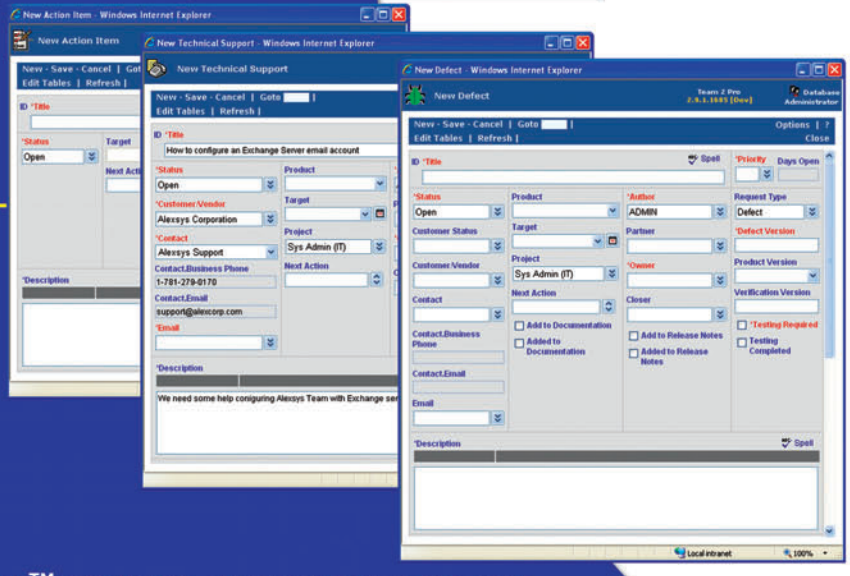
Team 2 Features

- 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.

Also notice that the FluentCassandra API offers the "LastError" property, which contains a reference to the last exception thrown out of the database. This can be useful to check when the state of the database isn't known already (such as when returning out of a set of calls that might have eaten the exception thrown, or if the database is configured to not throw exceptions).

Once Again, with Feeling

Connecting to the database, creating the keyspace (and later dropping it), defining the column families and putting in some seed data—I'm probably going to want to do these things a lot within these tests. That sequence of code is a great candidate to put into pre-test setup and post-test teardown methods. By dropping the keyspace after and recreating it before each test, I keep the database pristine and in a known state each time I run a test, as shown in **Figure 5**. Sweet.

'Look Upon My Works, All Ye Mighty, and Despair!'

Reading data from Cassandra takes a couple of forms. The first is to fetch the data out of the column family using the Get method on the CassandraColumnFamily object, shown in **Figure 6**.

This is great if I know the key ahead of time, but much of the time, that's not the case. In fact, it's arguable that most of the time, the exact record or records won't be known. So, another approach (not shown) is to use the FluentCassandra LINQ integration to write a LINQ-style query. This isn't quite as flexible as traditional LINQ, however. Because the column names aren't known ahead of time, it's a lot harder to write LINQ queries to find all the Newards (looking at the LastName name/value pair in the column family) in the database, for example.

Fortunately, CQL rides to the rescue, as shown in **Figure 7**.

Figure 7 **Using Cassandra LINQ Integration to Write a LINQ-Style Query**

```
[TestMethod]
public void StoreAndFetchSomeDataADifferentWay()
{
  using (var db = new CassandraContext(keyspace: "Earth", server: Server))
  {
    var peopleCF = db.GetColumnFamily("People");
    Assert.IsNotNull(peopleCF);

    Assert.IsNull(db.LastError);

    dynamic charlotte = peopleCF.CreateRecord("CharlotteNeward");
    charlotte.FirstName = "Charlotte";
    charlotte.LastName = "Neward";
    charlotte.Gender = "F";
    charlotte.Title = "Domestic Engineer";
    charlotte.RealTitle = "Superwife";

    db.Attach(charlotte);
    db.SaveChanges();

    Assert.IsNull(db.LastError);

    var newards =
      db.ExecuteQuery("SELECT * FROM People WHERE LastName='Neward'");
    Assert.IsTrue(newards.Count() > 0);
    foreach (dynamic neward in newards)
    {
      Assert.AreEqual(neward.LastName, "Neward");
    }
  }
}
```

Note, however, that if I run this code as is, it will fail—Cassandra won't let me use a name/value pair within a column family as a filter criteria unless an index is defined explicitly on it. Doing so requires another CQL statement:

```
db.ExecuteNonQuery(@"CREATE INDEX ON People (LastName)");
```

> Notice the use of the "dynamic" facilities of C# 4.0 to reinforce the idea that the column family is not a strictly typed collection of name/value pairs.

Usually, I want to set that up at the time the column family is created. Note as well that because Cassandra is schema-less, the "SELECT *" part of that query is a bit deceptive—it will return all the name/value pairs in the column family, but that doesn't mean that every record will have every column. This means, then, that a query with "WHERE Gender='F'" will never consider the records that don't have a "Gender" column in them, which leaves Rick, Ted and "The Artist Formerly Known as Prince" out of consideration. This is completely different from a relational database management system, where every row in a table must have values for each and every one of the columns (though I often duck that responsibility by storing "NULL" in those columns, which is considered by some to be a cardinal sin).

The full CQL language is too much to describe here, but a full reference is available on the Cassandra Web site at bit.ly/MHcWr6.

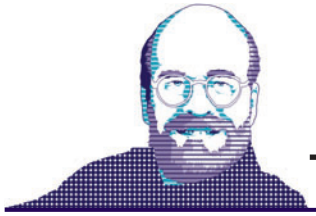
Wrapping up, for Now

I'm not quite done with the cursed prophetess just yet—while getting data in and out of Cassandra is the most interesting part to a developer (as that's what they do all day), multi-node configuration is also a pretty big part of the Cassandra story. Doing that on a single Windows box (for development purposes; you'll see how it would be easier to do across multiple servers) is not exactly trivial, which is why I'll wrap up the discussion on Cassandra by doing that next time.

For now, happy coding! ■

TED NEWARD *is an architectural consultant with Neudesic LLC. He has written more than 100 articles and authored or coauthored a dozen books, including "Professional F# 2.0" (Wrox, 2010). He is an F# MVP and noted Java expert, and speaks at both Java and .NET conferences around the world. He consults and mentors regularly—reach him at ted@tedneward.com or Ted.Neward@neudesic.com if you're interested in having him come work with your team. He blogs at blogs.tedneward.com and can be followed on Twitter at Twitter.com/tedneward.*

THANKS to the following technical expert for reviewing this article: *Kelly Sommers*



CHARLES PETZOLD

Viewing the Night Sky on Your Windows Phone

In the opening pages of his classic history, "The Copernican Revolution" (Harvard University Press, 1957), Thomas S. Kuhn evokes a world many centuries ago in which people had an intimate familiarity with the night sky. Not yet subject to the persistent illumination of artificial lighting, these observers experienced firsthand how the dome of stars seems to rotate during the night in a long arc around the North Star, and how the configuration of stars at sunrise and sunset progresses through a yearly cycle of 12 steps of the zodiac.

They also observed how several stars did not follow this same pattern. These oddities seemed to wander through the panorama of fixed stars, sometimes even reversing motion relative to the celestial dome. These exceptions to the general rule are still known by the ancient Greek word for wanderer: πλανήτης, or planet.

The attempt to trace and predict the motion of these planets was the beginning of the most famous upheaval in the history of science, eventually leading to the overthrow of primitive cosmology and the displacement of the Earth from the center of the universe. At the same time that the human race gained a much better understanding of its place in the cosmos, it also began to lose that special relationship with the night sky.

Using AstroPhone

These days most of us have not the time, patience or quietness of mind to stare at the night sky long enough to get a feel for its cyclical patterns or detect the wandering of planets. Instead we rely on star charts or computer-based astronomical tools to help out.

This column describes my own modest contribution to the genre, a little program for Windows Phone with the silly name AstroPhone. The downloadable source code for the application consists of an XNA program called AstroPhone and a library named Petzold.Astronomy that contains much of the data and number crunching.

This program has no UI whatsoever except the effort required to point the back of the phone toward a location in the night sky. The phone's screen then displays the stars, constellations and planets located there. **Figure 1** shows a typical screen, made at 3:45 p.m. local time on July 15, 2012, in the region of New York City, with the phone held in a westerly direction.

The green line is the horizon labeled with compass points. (Notice the W for west.) The red line is the ecliptic, which is the plane through which all the planets approximately orbit, with tick marks every 15°. You can see Jupiter and Venus due to set within the next 90 minutes or so. Also visible—on the screen if not in the mid-afternoon sky—is the constellation of Taurus.

To position the stars and planets correctly on the screen, the program needs to combine information from several sources:

- Data and algorithms to derive the position of the stars and planets (described in the following sections).
- The current date and time to use in those algorithms.
- The phone's GPS so the program knows where the phone is located on the surface of the Earth.
- The phone's Motion sensor so the program knows how the phone is oriented in 3D space, and hence what part of the celestial sphere it's pointed toward.

Due to its heavy reliance on the phone's sensors, this program will not run on the Windows Phone emulator.

These days most of us have not the time, patience or quietness of mind to stare at the night sky long enough to get a feel for its cyclical patterns or detect the wandering of planets.

In several previous installments of this column I've been discussing some of the concepts of using the Motion sensor and converting that information to various coordinate systems. Those earlier articles are all available on the *MSDN Magazine* Web site (bit.ly/NoMc8R). Much of my approach to positional astronomy is based on the book by Jean Meeus, "Astronomical Algorithms" (Willmann-Bell, 1998), although I've sometimes used somewhat simplified calculations, and I make no guarantees that I've eliminated all errors.

Stars and Constellations

If you look up a particular star on Wikipedia, you'll find its location is given in terms of two numbers—a right ascension and a declination. For example, Betelgeuse has a right ascension of 5 hours,

Code download available at archive.msdn.microsoft.com/mag201210TouchAndGo.

55 minutes, 10.3053 seconds, and a declination of 7°, 24 minutes, 25.426 seconds. This position is relative to the equatorial coordinate system, in which the Earth's equator divides the universe into positive angular declinations to the north and negative angular declinations to the south. (Polaris, also known as the North Star, has a declination of very nearly 90°.)

Traditionally, right ascension is measured in hours, with each hour equivalent to 15°. A right ascension of zero corresponds to the same direction as north at midnight local time on the date of the vernal equinox. Because the plane of the Earth's equator remains roughly constant throughout the year (and throughout the centuries), and because the stars are so far away, they are considered to have the same equatorial coordinates regardless of the position of the Earth. These equatorial coordinates for stars are specified for a particular epoch (for example, the year 2000), and small annual adjustments can be applied to account for the slight motion of the stars.

Many star catalogs exist, most of them with many more stars than I wanted to display in my program. I decided to use the updated version of a catalog that originated more than 100 years ago: the Bright Star Catalog, 5th revised edition, which I obtained from an FTP site maintained by the Strasbourg Astronomical Data Center (bit.ly/T51GE5).

Even this catalog provided much more information than I needed, and also somewhat less. For the most part I wanted my program to display only the major stars associated with the 88 standard constellations used in modern astronomy. That was easy enough because the stars have designations (first developed by Johann Bayer more than 400 years ago) that indicate which constellation the star belongs to with a three-letter abbreviation and a unique Greek letter.

But I also wanted to enhance each constellation with those familiar line connections between the stars so that Leo would actually look like a stick-figure lion. The lines connecting the stars in the constellations are not standard, and I didn't know of a star catalog that included them.

So I added them myself. I wrote a little Windows Presentation Foundation (WPF) program that accessed the Bright Star Catalog and displayed the stars of each of the 88 constellations. The program responded to mouse clicks on pairs of stars and recorded the results. Generally I based the connecting lines on diagrams of the constellations on Wikipedia, which are credited to the International Astronomical Union and *Sky & Telescope* magazine.

The WPF program consolidated the data into a file named Constellations.xml, which you can find in the Data directory of the Petzold.Astronomy library project. Each Constellation tag contains a collection of Star tags. Each star is identified with a number from the Bright Star Catalog. The Star tags in each constellation are then followed by a collection of Connector tags that define lines between pairs of numbered stars.

Figure 1 **A Typical AstroPhone Screen**

The Planets

As even primitive stargazers realized, the location of the planets is much more complex than that of the stars. The planets have elliptical orbits around the sun, but irregularities result from mutual gravitational interaction. An algorithm to determine the location of a planet at a particular time is often known as a "theory," and generally based on a Fourier series.

The planetary theory I used is called VSOP87, *Variations Séculaires des Orbites Planétaires*, 1987 version, which is maintained by the Bureau des Longitudes in Paris. The files containing the data of a portion of VSOP87 are also included in the Data directory of the Petzold.Astronomy project. The VsopCruncher class performs the calculations for each planet for a particular point in time.

The locations of the planets are usually calculated in ecliptic coordinates, but with the sun in the center—that is, heliocentric rather than geocentric. For each planet, these coordinates consist of:

- an ecliptic longitude that goes from 0° to 360° as the planet makes a complete orbit around the sun during its particular year
- an ecliptic latitude that's close to zero because all the planets orbit the sun in roughly the same plane
- a radius, which is the distance of the planet from the sun.

These coordinates are great for plotting the movement of planets around the sun. However, if you want to calculate the position of the planet as viewed from Earth, it's convenient to convert these spherical coordinates into three-dimensional rectangular coordinates, where a planet's position is indicated by X, Y and Z values with the sun occupying the center origin at (0, 0, 0). By subtracting the rectangular coordinate of any planet from the rectangular coordinate of the Earth, you get a 3D vector from the Earth to that planet, which can then be converted into the right ascension and declination values of a geocentric ecliptic coordinate—the same coordinate system used for the stars.

The moon's movement is much more complex than that of the stars and planets, and has to be handled as a separate case. I chose not to include the moon in this version of the AstroPhone program. Besides, if I were to include the moon, I'd feel obliged to show the current phase, and that would complicate the job even more.

Chains of Calculations

To perform the necessary calculations in a methodical way, I defined a hierarchy of classes:

```
CelestialBody
        Constellation
        Star
        SolarSystemBody
                Sun
                Planet
                        Earth
```

The Star and Constellation classes also play double roles as deserialization targets of the Constellations.xaml file.

Figure 2 **The Parent CelestialBody Class**

```
public abstract class CelestialBody
{
  // Only used internally
  private GeographicCoordinate Location { set; get; }

  // Set here, used by descendant classes
  protected Time Time { private set; get; }

  // Set by descendant classes, used here
  protected EquatorialCoordinate EquatorialCoordinate { set; private get; }

  // Set here, used external to library
  public HorizontalCoordinate HorizontalCoordinate { private set; get; }

  // Used externally to retain screen location
  public float ScreenX;
  public float ScreenY;

  // Called external to update HorizontalCoordinate
  public void Update(Time time, GeographicCoordinate location)
  {
    bool needsUpdate = false;

    if (!this.Time.Equals(time))
    {
      this.Time = time;
      needsUpdate = true;
      OnTimeChanged();
    }

    if (!this.Location.Equals(location))
    {
      this.Location = location;
      needsUpdate = true;
    }

    if (needsUpdate)
    {
      this.HorizontalCoordinate =
        HorizontalCoordinate.From(this.EquatorialCoordinate,
                                  this.Location, this.Time);
    }
  }

  // Overridden by descendant classes to update EquatorialCoordinate
  protected virtual void OnTimeChanged()
  {
  }
}
```

As shown in **Figure 2**, the CelestialBody class is responsible for computing a HorizontalCoordinate from an EquatorialCoordinate based on the geographic location of the phone and the current date and time. The descendant classes calculate that EquatorialCoordinate property in overrides of the OnTimeChanged method.

The SolarSystemBody class shown in **Figure 3** calculates the EquatorialCoordinate property from a HeliocentricLocation vector provided by its descendant classes, and the Planet class, shown in **Figure 4**, calculates that HeliocentricLocation based on a call to the VsopCruncher.

The Star class is even simpler because it can calculate an EquatorialCoordinate with just a small adjustment to its year 2000 position.

The XNA Front End

I began writing the AstroPhone program using Silverlight, which worked great with the first several constellations. But the more constellations I added, the slower it became.

Keep in mind that when using Silverlight for such a program, all the text, lines and dots are TextBlock, Line and Path elements in a Panel of some sort. To get fluid motion as you sweep the phone in an arc, this needs to be just one Panel, and it's got to have everything in it. Performance degrades even more if you start removing items or making them invisible when they're not in view.

I spent a lot of time trying to improve the performance of this Silverlight program. I discovered that using a Canvas worked better than a single-cell Grid. I found that when displaying a lot of Line elements, you could improve performance by setting one coordinate of the line to the point (0, 0), adjusting the other coordinate by the same amount and using a TranslateTransform to move it into position.

But with all 88 constellations in place, the video frame rate dropped down to a single frame per second, and there was simply no alternative except to abandon Silverlight. That's why AstroPhone is an XNA program.

Interestingly, the slowest part of AstroPhone isn't graphics at all. It's the deserialization of the Constellations.xml file, which occurs in the OnActivated override of the Game class. The program then builds several collections for updating and rendering the celestial objects. The primary collection used for updating coordinates is called celestialBodies. As the name suggests, this is a collection of instances of classes that derive from CelestialBody, and it contains 832 objects—735 objects of type Star, 88 of type Constellation, one Sun and all eight Planet objects. (Pluto is not included in VSOP87.)

In an XNA program for Windows Phone, both the Update and Draw overrides in the Game class are called at the rate of 30 times per second. Update is responsible for obtaining input from the sensors (GPS and Motion, in this case) and preparing data for the Draw method.

To achieve the smoothest response to movements of the phone, the Update override loops through the entire celestialBodies collection, obtains the HorizontalCoordinate property from each

Figure 3 **The SolarSystemBody Class**

```
public class SolarSystemBody : CelestialBody
{
  protected SolarSystemBody()
  {
  }

  protected SolarSystemBody(string name, Color color)
  {
    this.Name = name;
    this.Color = color;
  }

  public string Name { protected set; get; }

  public Color Color { protected set; get; }

  // Set by descendant classes, used here
  protected Vector3 HeliocentricLocation { set; private get; }

  protected override void OnTimeChanged()
  {
    // Calculate geocentric coordinates
    Vector3 bodyLocation = this.HeliocentricLocation -
                           Earth.Instance.HeliocentricLocation;
    EclipticCoordinate geocentricCoordinate =
      new EclipticCoordinate(bodyLocation);
    this.EquatorialCoordinate =
      EquatorialCoordinate.From(geocentricCoordinate, this.Time);

    base.OnTimeChanged();
  }
}
```

Figure 4 **The Planet Class**

```
public class Planet : SolarSystemBody
{
  VsopCruncher vsop;

  public Planet(string strPlanetAbbreviation, string name, Color color) :
    this(strPlanetAbbreviation)
  {
    this.Name = name;
    this.Color = color;
  }

  protected Planet(string strPlanetAbbreviation)
  {
    vsop = new VsopCruncher(strPlanetAbbreviation);
  }

  protected override void OnTimeChanged()
  {
    Angle latitude = Angle.FromRadians(vsop.GetLatitude(this.Time.Tau));
    Angle longitude = Angle.FromRadians(vsop.GetLongitude(this.Time.Tau));
    double radius = vsop.GetRadius(this.Time.Tau);

    this.HeliocentricLocation =
      new EclipticCoordinate(longitude, latitude, radius).RectangularCoordinates;

    base.OnTimeChanged();
  }
```

object, and uses the current Motion matrix to convert that to a two-dimensional point on the screen, which it then stores in the ScreenX and ScreenY properties of the CelestialBody object. The Draw method then accesses those ScreenX and ScreenY properties to draw the object on the screen.

But that calculation is solely to account for the movement of the screen. It's also necessary for each CelestialBody object to periodically update its HorizontalCoordinate property as time passes and both the Earth and other planets move a bit. Still, this update isn't crucial to the smooth operation of the program. The HorizontalCoordinate property is based on the current date and time and the geographic location of the user, but neither of these items changes quickly enough to affect the position of the stars and planets in the short term.

For this reason, the Update override of the Game class deals with the Update methods of the CelestialBody objects in a leisurely manner. Only one item in the celestialBodies collection is updated for each call of the Game class's Update override, requiring a cycle of about 28 seconds to loop through the entire collection of 832 objects.

For rendering purposes, other collections are maintained because different types of celestial objects are rendered in different ways. The visibleStars collection contains the 735 Star objects that are rendered on the screen, the constellations collection has the 88 constellations, and the systemBodies collection has the sun and the seven planets, excluding Earth.

Like every other class that derives from CelestialBody, the Constellation class sets an EquatorialCoordinate property, but this is solely for purposes of positioning the name of the constellation. That position is calculated by each Constellation instance by taking an average of the connected stars that make up the constellation.

The connection lines themselves were rather tricky. Each Constellation object has a collection of Connector objects, each of which references a pair of stars in the constellation. But these Connector objects come from the original Constellations.xml file, and they reference the pair of connected stars with ID numbers. To quicken the line drawing, the program spends part of the initialization process supplementing each pair of ID numbers with a StarConnector object, which is a pair of actual Star objects. Thus, the program can draw the connector lines by referencing the ScreenX and ScreenY properties of the actual Star objects.

Figure 5 **The Draw Method to Render Constellations and Stars**

```
protected override void Draw(GameTime gameTime)
{
  // Dark blue sky
  GraphicsDevice.Clear(new Color(0, 0, 0x20));

  spriteBatch.Begin(SpriteSortMode.Immediate, null, null, null, null, null,
    displayMatrix);
  ...

  // Loop through constellations
  foreach (Constellation constellation in constellations.ConstellationList)
  {
    if (!float.IsNaN(constellation.ScreenX))
    {
      // Display constellation name
      Vector2 textSize = labelFont.MeasureString(constellation.LongName);
      spriteBatch.DrawString(labelFont, constellation.LongName,
        new Vector2(constellation.ScreenX - textSize.X / 2,
        constellation.ScreenY - textSize.Y / 2), Color.Gray);
    }

    // Display constellation connectors
    if (constellation.StarConnectors != null)
    {
      foreach (StarConnector starConnector in constellation.StarConnectors)
      {
        Vector2 point1 = new Vector2((float)starConnector.From.ScreenX,
          (float)starConnector.From.ScreenY);
        Vector2 point2 = new Vector2((float)starConnector.To.ScreenX,
          (float)starConnector.To.ScreenY);

        if (!float.IsNaN(point1.X) && !float.IsNaN(point2.X))
          lineRenderer.Draw(spriteBatch, point1, point2, 1, Color.White);
      }
    }
  }

  // Now display the stars themselves
  foreach (Star star in visibleStars)
    if (!float.IsNaN(star.ScreenX))
      starDotRenderer.Draw(spriteBatch,
        new Vector2(star.ScreenX, star.ScreenY), Color.White);
  ...

  spriteBatch.End();
  base.Draw(gameTime);
}
```

Figure 5 shows the portion of the Draw method that renders the constellation names, the connecting lines and the stars themselves.

Although programs such as AstroPhone provide a helpful guide to the stars and planets, nothing comes close to the experience of actually looking at them in real life. Perhaps as people gain more knowledge with the help of programs such as this, they will once again develop an intimate familiarity with the night sky. ■

CHARLES PETZOLD *is a longtime contributor to* MSDN Magazine*, and is currently updating his classic book, "Programming Windows" (Microsoft Press, 1998), for Windows 8. His Web site is charlespetzold.com.*

THANKS to the following technical expert for reviewing this article: *Donn Morse*

Brain Droppings

Over the course of a year many ideas occur to me that are important, but aren't substantial enough to merit a full column. I've collected these thoughts here, in the inaugural edition of what will be my annual "Brain Droppings" column. I plan to make this a pillar of my editorial year (the other being, of course, my April Fool's Day column). Use the comment feature on the Web site to tell me which of the following ideas resonate with you.

Many computer games display the elapsed time on their victory screens, for example, "Congratulations, you solved the puzzle in 12 minutes 15 seconds." Suppose instead it said, "Congratulations, you just wasted 12:15 of your life that you'll never get back again. Nice work, doofus."

> Anything worth doing is worth overdoing. And anything worth fighting for is worth fighting dirty for.

Microsoft Word didn't recognize the word "doofus" in the previous sentence.

The mechanical hard disk is now dead. Solid-state drives rule. But do we still have to call them disks, even though they're not circular anymore?

I was a physics major as an undergraduate, so I'm following the Higgs boson discovery with some interest. I once asked a high-energy physicist what he'd do when they finally found everything in the atom. He thought a second, then said: "First, we have one *hell* of a party. And then we all go look for new jobs."

I've had enough. The next time I sit in a presentation and the speaker just reads off his PowerPoint bullets, I'm going to call him on it. In the middle of his session, publicly and brutally. Make sure that speaker isn't you. Read my March 2011 column (msdn.microsoft.com/magazine/gg650665) for starters.

It should not take four Windows processes to run a graphics adapter, but Intel on my ThinkPad does: igfxpers, igfxtray, hkcmd, igfxsrvc. Guys, get with it, OK?

Here are two things I teach my daughters: Anything worth doing is worth overdoing. And anything worth fighting for is worth fighting dirty for.

When Microsoft Word upgraded versions some time ago, the desktop icon changed from a freestanding light-blue W to a dark-blue W with a box around it. An IT director once told me: "I have 60,000 users in my admin space. Do you have any idea how long it takes and how much it costs me to tell 60,000 people that the thing they used to get from the light-blue W they now get from the dark-blue W with the box around it? I wish I could send Microsoft the bill for my cost of just that one change."

I cringe every time I see a new technology intended for use by the driver of a car. It's not just the Ford SYNC that runs on Microsoft Windows Embedded Automotive ("Would you like today's horoscope? What's your sign?" My reply: "Caution, Merging Traffic"). This is a problem everywhere. How about we use today's technology to help the driver do a better job of not killing himself and others, instead of distracting him even more? Like short-circuiting the driver's cell phone so it doesn't ring, and automatically replying, "Sorry, he's busy driving now. I'll have him call you as soon as he stops." In fact, every design meeting on automotive technology should close with the question, "Are the design decisions we've just made going to kill more people or fewer?"

The official kilogram seems to be losing mass (see nyti.ms/U9KFcl). I'd be lying if I said I was losing a lot of sleep over this.

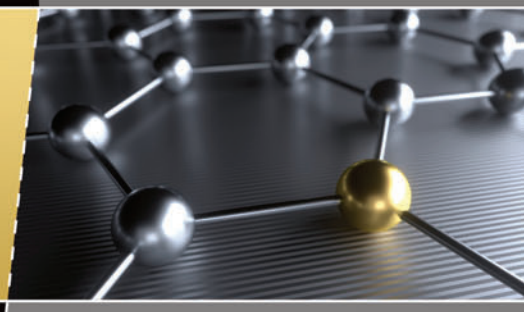
If you say, "We'll just train the users," you're barking up the wrong tree. Go back and figure out how to build your app so that it will just work.

I'd like to remove the function MessageBox from the Windows API. It's just too easy to pop up a box saying, "Error 15. You're toast. [OK]," instead of making the effort to explain the problem in terms of the user's mental model, not the program's implementation model. Better yet, figure out how to avoid that error in the first place. ■

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.*

Powerful Tools for Developers

New! V4.6



Create & Edit PDFs in .Net - ActiveX - WinRT

- Edit, process and print PDF 1.7 documents programmatically.
- Fast and lightweight 32 and 64-bit components for .Net, COM and WinRT applications.
- Create, fill-out and annotate PDF forms.

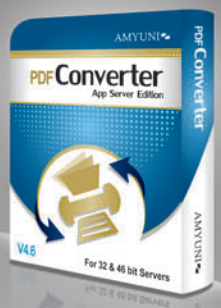
Complete Suite of Accurate PDF Components

- All your PDF processing, conversion and editing in a single package.
- Combines Amyuni PDF Converter and PDF Creator for easy licensing, integration and deployment.
- Includes our Microsoft certified PDF Converter printer driver.
- Export PDF documents into other formats such as JPeg, Word, Excel or Silverlight.

Advanced HTML to PDF & XAML

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

High Performance PDF Printer Driver



- Our high-performance printer driver optimized for Web, Application and Print Servers. Print to PDF in a fraction of the time needed with other tools. WHQL tested for Windows 32 and 64-bit including Windows Server 2008 and Windows 8.
- Standard PDF features included with a number of unique features. Interface with any .Net or ActiveX programming language.
- Easy licensing and deployment to fit system administrator's requirements.

Windows 7 | CERTIFIED FOR Windows Server 2008 | Ready for Windows 8

AMYUNI

All development tools available at

www.amyuni.com

USA and Canada
Toll Free: 1866 926 9864
Support: 514 868 9227
sales@amyuni.com

Europe
UK: 0800-015-4682
Germany: 0800-183-0923
France: 0800-911-248