THE MICROSOFT JOURNAL FOR DEVELOPERS

AUGUST 2012 VOL 27 NO 8

msdn magazine

COLUMNS

Microsoft

Start a Revolution

Refuse to choose between desktop and mobile.

With the brand new NetAdvantage for .NET, you can create awesome apps with killer data visualization today, on any platform or device.

Get your free, fully supported trial today!

www.infragistics.com/NET

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


PopulationExplosion
VISUALIZING THE GROWTH OF THE WORLD COMMUNITY
XamPivotGrid
XamDataChart
Population
Measures
Country
Albania
Austria
Belarus
Belgium
Bosnia and Herzegovina
Bulgaria
Croatia
Denmark
Estonia
Finland
France
Germany
Greece
Hungary
Iceland
Ireland
Italy
Latvia
Lithuania
Luxembourg
Macedonia, FYR
Moldova
Montenegro
Netherlands
List
Maria Anders
Ana Trujillo
Antonio Moreno
Thomas Hardy
Christina Berglund
Hanna Moos
Frédérique Citeaux
Martin Sommer
HTML
Compatible with
Microsoft® Visual Studio® 2012

msdn magazine

AUGUST 2012 VOLUME 27 NUMBER 8

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

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

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

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

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

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

Microsoft

BPA WORLDWIDE

Printed in the USA

LEADTOOLS®
MEDICAL SDKs

DICOM SDK / PACS SDK

3D VOLUME RECONSTRUCTION

WEB VIEWER FRAMEWORK / WORKSTATION FRAMEWORK

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

MEDICAL IMAGE PROCESSING / ANNOTATIONS

C++ .NET METRO 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

MICHAEL DESMOND

On Point with Julie Lerman

In the world of software development, nothing is certain except death, taxes and data. As *MSDN Magazine* Data Points columnist Julie Lerman points out: "Nobody can avoid data, so it's a topic that's important to everyone."

Which is why Data Points has been a fixture in *MSDN Magazine* since John Papa first penned the column back in 2002. Papa launched the column to address what he felt was a shortage of data-related coverage in the magazine (and the industry in general). When he stepped away from the column in 2009 after taking a job with Microsoft, Papa recommended Julie Lerman to take his spot.

I asked Lerman about her experience writing the column and about her thoughts on data and development. As Lerman told me, inspiration is not hard to find.

"I've tried to use the column to explain things that either I'm curious or confused about, such as what the heck NoSQL is," she says. "Or as a way to share answers to questions that I'm asked frequently—for example, about Entity Framework [EF]."

Michael Desmond: The .NET data space has been anything but boring, with plenty of infrastructure work pulling developers in different directions. Have things settled down? Any advice for developers trying to make big-picture decisions about data in 2012?

Julie Lerman: Settled? Ha! I think they're moving faster. I do think the ORMs [object-relational mappers] are going to settle in as the "classics" for a while, and Entity Framework seems to have become the standard for out-of-the-box .NET data access.

There's so much innovative work and thinking going on, especially with the focus on big data, NoSQL and CQRS [Command Query Responsibility Segregation], to name a few. But not everyone has to work with the vast amounts of data that comes under that umbrella. I really do think ORM over relational database is the new norm.

How well has EF come together over its brief, if turbulent, run from the first version in 2008 through the EF4 iterations the last couple years, and finally to EF5 today?

EF started as a project of some serious, data-wizard, Ph.D.-level folks at Microsoft Research. But for developers it seemed to be only a partial solution. The addition of POCO [Plain Old CLR Object] support in EF4 was a huge leap forward, and then Code First plus the much-simpler-to-use API [DbContext] changed the broader perception of EF dramatically. It wasn't until those last two pieces arrived that people at Microsoft who are outside of the data team started paying attention to it. And the developers' eyes, enthusiasm and excitement followed.

It's also been fascinating to witness the evolution of how the EF team at Microsoft works. After some challenging interactions with a voluble community that was unhappy with the initial release, they responded with some fantastic changes that made EF palatable to a much wider audience. Now the entire team is focused on responding to community feedback.

I get the feeling that the .NET dev community—the subset that engages with new releases and providing feedback—has a great sense of ownership of EF and is now very supportive of the work the team is doing.

How do you find time to write consistently given the demands of your day job? How does writing help you improve as a developer?

You know better than anyone that my articles are rarely delivered on time. I just steal the time from other tasks and deadlines, so I'm constantly juggling. I also have to steal some of that time from my personal life. I think the fact that my husband and I don't have kids makes that a bit easier.

The process of writing does indeed have a positive effect on my development skills. I'm reluctant to write something down until I've explored it inside and out, which forces me to learn even more deeply something I may already have a great deal of comfort with. I think we have a great responsibility not to misdirect people who depend on us for their knowledge. I'm constantly questioning what I know and how I do things, and I sometimes very reluctantly drag myself through some process of evolution. But it always has its rewards.

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

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

Try the new **real-time visual dashboard** in **OnTime Scrum.**

OnTime Scrum

Agile project management & bug tracking

- product backlogs, releases, and sprints
- powerful user story and bug tracking
- automated burndown charts
- **real-time visual project dashboard**

Enterprise quality software at prices any team can afford.

$7 per user per month

for teams of 11+ users

$10 per month for up to 10 users

special small-team pricing

OnTime Dev Suite

Also available from the **OnTime Dev Suite**:

- **OnTime Help Desk** — Customer support for software apps
- **OnTime Team Wiki** — Project wiki for colloborative dev teams

Visit **OnTimeNow.com/MSDN** for these 3 great resources:

1. Watch our popular intro to Scrum video

2. Download and print our free Scrum Diagram

Free Trial

3. Get started free with OnTime Dev Suite

axosoft

axosoft.com • @axosoft • 800.653.0024

DINO ESPOSITO

Mobile Site Development, Part 3: Routing Requests

More often than not, a mobile site is the subset of a larger site built for a desktop audience. A desktop user often visits your site using, for example, a laptop with a high screen resolution and significant computing power. In addition, the user relies on stable connectivity and isn't particularly concerned about draining the battery. All these parameters make a huge difference for architects and developers of Web sites, as well as domain experts. When it comes to mobile sites, the hardest part is not coding but figuring out the use cases to code—and before that, the business cases for which you want to have a mobile site (or a mobile application).

The mobile site exists to make it easier for users on the go to consume the most relevant services that you already expose through the main Web site. So let's assume we have a well-defined set of use cases and we're ready to start coding and producing some good markup. But, first and foremost, how do we reach the mobile site?

I firmly believe that a mobile site should be a standalone site that corresponds to a distinct IIS application. This greatly simplifies the development experience. You focus only on mobile use cases. You optimize rendering and the application layer only for mobile use cases. You deal only with technologies and tools related to mobile scenarios. Finally, you test it more easily.

> When it comes to mobile sites, the hardest part is not coding but figuring out the use cases to code.

On the other hand, as a user, I hate having to type a distinct URL just to view the mobile version of a site in which I'm interested. Wouldn't it be great if you could just type or bookmark www.contoso.com and let the site figure out whether you're coming from a mobile device or a laptop? Once your origin is ascertained, the site could possibly redirect you to the mobile site. Applying such a strategy can deliver a nice experience, but this requires some assumptions and having some machinery in place.

Routing Users to the Right Site

Let's assume there are two distinct Web sites in place—say, www.contoso.com and m.contoso.com. The user, however, isn't expected to know about the m-site; she only knows about the main Web site. So she types www.contoso.com into her mobile device. Some code hosted in the main site intercepts the request and examines the user agent string. If the request is coming from a desktop browser, nothing happens and the request proceeds as usual. If the requesting browser is hosted on a mobile device, the user is redirected to a landing page where she's asked to choose between the full site and the mobile site. **Figure 1** offers a graphical view of the strategy I'm outlining.

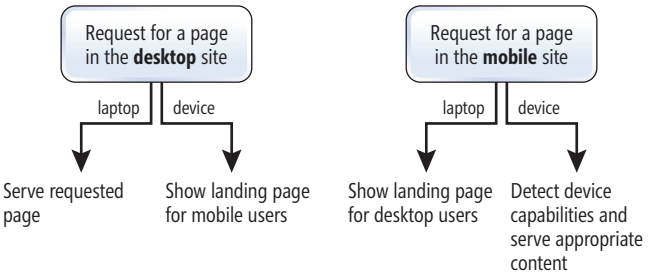


Figure 1 **A Strategy for Routing Users to the Most Appropriate Site**

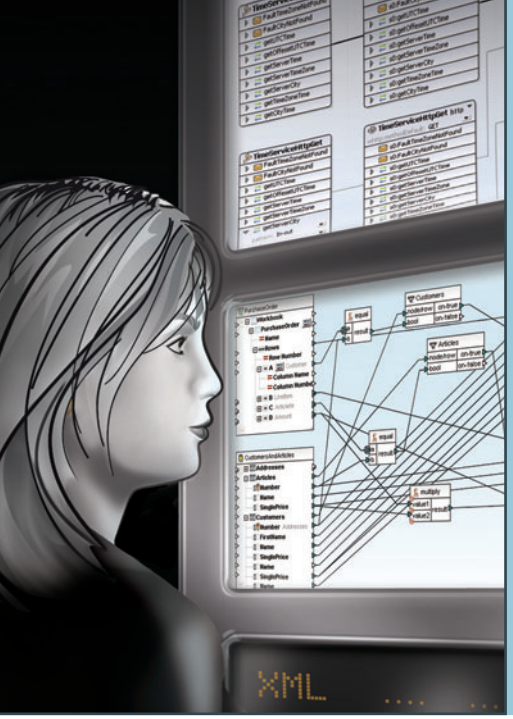
More generally, both sites have an interceptor that filters any incoming requests and redirects to a landing page if the requesting device isn't compatible with the type of the site. To keep things even simpler, you can also let laptops view mobile pages if explicitly requested and host only one landing page for when the main site is requested from a mobile device.

This strategy only explains what to do on the first request to a site—typically directed to the homepage. For successive requests that the user can make as she works with the site, you don't want the interceptor to kick in again and show the landing page. Let's see how to implement this strategy.

An HTTP Module to Route Requests

In ASP.NET, the standard way to intercept incoming requests is installing a HTTP module. For each intercepted request, the HTTP module will examine the user agent string and determine whether or not the browser is hosted on a mobile device. If the browser runs on a mobile device, the HTTP module redirects the user to a given landing page; if not, it will simply let the request pass and be processed as usual. **Figure 2** shows the source code of a sample HTTP module for routing mobile requests to a (mobile) landing page.

There are basically three use-case scenarios. One is when the user with a mobile device reaches a landing page and confirms she wants to view the full site. Another scenario is when the mobile user requests a page on the full site because she's following a link in one of the full-site pages. Put another way, the mobile user received the landing page, confirmed she wants to view the full site



Connect legacy technologies affordably with the complete set of data integration tools from Altova®

ALTOVA® missionkit®

Experience how the Altova MissionKit®, the integrated suite of XML, data mapping, and database tools, can help you leverage existing technology and business software investments while integrating modern technologies – without breaking your budget.

NEW in Version 2012:

- Streaming reading/writing of files for large-scale ETL applications
- Sorting of data mapping results
- New data processing functions
- JDBC drivers (in addition to ADO and ODBC)
- New, native Java APIs

The Altova MissionKit includes multiple intelligent tools for data integration:

MapForce® – Graphical data mapping, transformation, & conversion tool

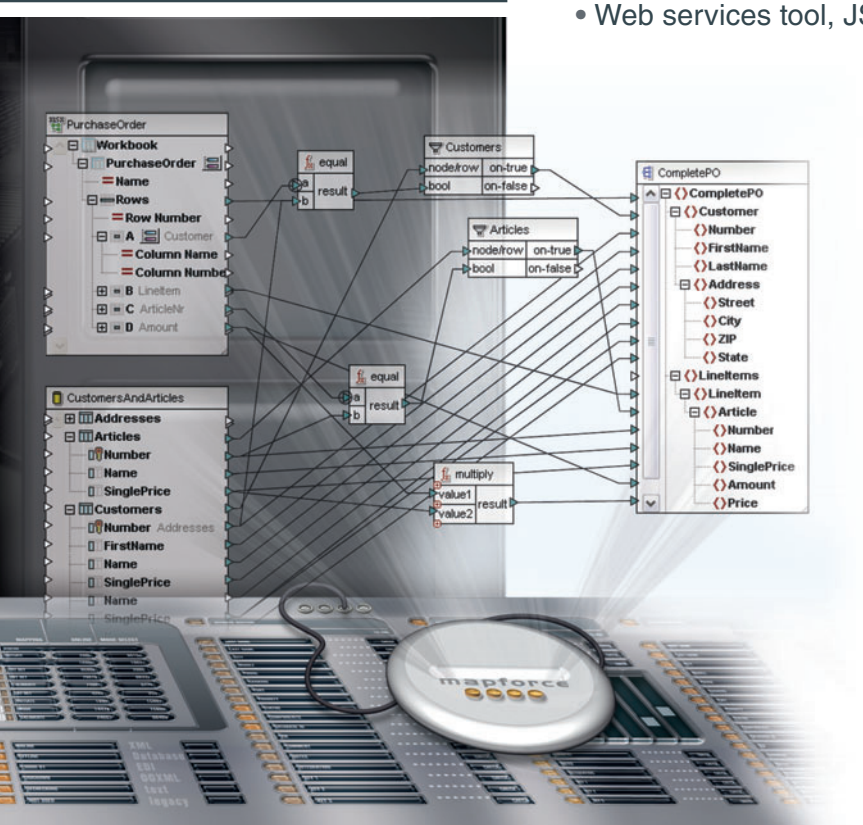
- Drag-and-drop data conversion with instant transformation & code gen
- Support for mapping XML, DBs, EDI, Excel®, XBRL, flat files & Web services

XMLSpy® – XML editor and Web services tool

- XML editor with strong database integration
- Web services tool, JSON <> XML converter

DatabaseSpy® – multi-database query, design & comparison tool

- Support for all major relational databases and translation between DB types
- SQL editor, graphical database design & content editor



Download a 30 day free trial!

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

Scan to learn more about data integration with the MissionKit.

Figure 2 **A Mobile Router Component Implemented as an HTTP Module**

```
public class MobileRouterModule : IHttpModule
{
  private const String ForceFullSiteCookieName = "FullSiteMode";
  public void Dispose()
  {
  }
 public void Init(HttpApplication context)
 {
   context.BeginRequest += OnBeginRequest;
 }

 private static void OnBeginRequest(Object sender, EventArgs e)
 {
   var app = sender as HttpApplication;
   if (app == null)
     throw new ArgumentNullException("sender");

   // Check whether it is a mobile site
   var isMobileDevice = IsMobileUserAgent(app);

   // The mobile user confirmed to view the desktop site
   if (isMobileDevice && ForceFullSite(app))
   {
     app.Response.AppendCookie(new HttpCookie(ForceFullSiteCookieName));
     return;
   }

   // The mobile user is navigating through the desktop site
   if (isMobileDevice && HasFullSiteCookie(app))
     return;

   // The mobile user is attempting to view a desktop page
   if (isMobileDevice)
     ToMobileLandingPage(app);
 }
}
```

and is now navigating through the site. Finally, the third scenario is when the mobile user is trying to visit the full site for the first time in that browser session. In the first two cases, the HTTP module lets the request pass. In the last case, it simply redirects the mobile user to an ad hoc landing page.

The landing page will be a mobile-optimized page that shows a message to the user and offers links to the homepage of the desktop site or mobile site. **Figure 3** shows a sample landing page. You can try it out live by pointing your own device to easycourt.net/contosoen.

The URL behind the "Mobile site" link points to the mobile site homepage. The other link points to the homepage of the full site with an extra query string parameter. The name and role of this parameter are up to you, but the name could be something like this: http://www.contoso.com?mode=full.

This parameter will be checked by the HTTP module through one of the functions mentioned in **Figure 2**, as shown here:

```
private static Boolean ForceFullSite(HttpApplication app)
{
  var full = app.Context.Request.QueryString["mode"];
  if (!String.IsNullOrEmpty(full))
    return String.Equals(full, "full", StringComparison.InvariantCultureIgnoreCase);
  return false;
}
```

You see in **Figure 2** that the user's choice is stored in a cookie. This means that a user who expressly chose to navigate to the full site with a mobile device won't be bothered any longer with a landing page as long as the cookie is available.

Before I discuss the structure of the landing page in more detail, let me specify how the HTTP module understands whether the user clicked to view the full site or is navigating through the full site. In **Figure 2** you see that the following code is used to check whether the cookie for viewing the full site is found:

```
private static Boolean HasFullSiteCookie(HttpApplication app)
{
  var cookie = app.Context.Request.Cookies[FullSiteModeCookie];
  return cookie != null;
}
```

If there's no cookie and no query string parameter, the user accessing the desktop site from a mobile device is simply redirected to the landing page:

```
private static void ToMobileLandingPage(HttpApplication app)
{
  var landingPage = ConfigurationManager.AppSettings["MobileLandingPage"];
  if (!String.IsNullOrEmpty(landingPage))
    app.Context.Response.Redirect(landingPage);
}
```

The key point is that the user might be redirected to the landing page only the first time she attempts to access a given desktop site from a mobile device. From that point on, any further requests have extra information that prevents the HTTP module from redirecting.

> I firmly believe that a mobile site should be a standalone site that corresponds to a distinct IIS application.

Detecting Mobile Devices

Of all the code you see in **Figure 2**, only one piece remains to fully explain: how you detect whether the requesting device is a mobile device. This can be achieved in different ways and with different levels of reliability. For example, you can simply rely on some code like that shown in **Figure 4**. It attempts to query the user agent string for some mobile-only keywords.

The list of keywords isn't exhaustive, but it's fairly representative of the mobile universe. This code may be extended at will to make it work better and better as new devices appear and as more mobile users start visiting your site. But that's precisely the point. Are you willing to constantly update this piece of code? Not having to revise such code is just one of the benefits that a Device Description Repository (DDR) offers. A DDR is a library built on top of a database that contains user agent strings. All a DDR does is parse the user agent and return a list of known capabilities. I said "known" and not "detected" because that's precisely the case.

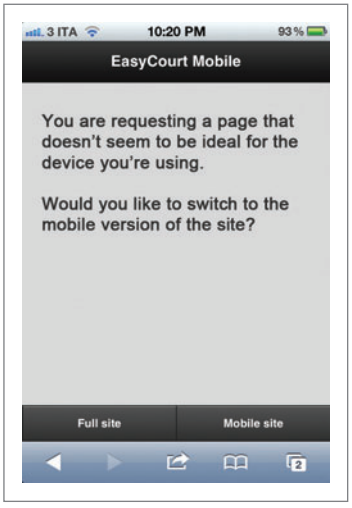


Figure 3 **A Sample Landing Page for a Mobile Site**

Figure 4 **Querying the User Agent String for Mobile-Only Keywords**

```
private static Boolean HasAnyMobileKeyword(String userAgent)
{
  string ua = userAgent.ToLower();
  return (ua.Contains("midp") ||
    ua.Contains("mobile") ||
    ua.Contains("android") ||
    ua.Contains("samsung") ||
    ua.Contains("nokia") ||
    ua.Contains("phone") ||
    ua.Contains("opera mini") ||
    ua.Contains("opera mobi") ||
    ua.Contains("blackberry") ||
    ua.Contains("symbian") ||
    ua.Contains("j2me") ||
    ua.Contains("windows ce") ||
    ua.Contains("vodafone") ||
    ua.Contains("ipad;") ||
    ua.Contains("maemo") ||
    ua.Contains("palm") ||
    ua.Contains("fennec") ||
    ua.Contains("wireless") ||
    ua.Contains("htc") ||
    ua.Contains("nintendo") ||
    ua.Contains("zunewp7") ||
    ua.Contains("silk");
}
```

A DDR gets its information statically from a frequently updated database. It doesn't detect capabilities on the device. However, this is an advantage of DDR solutions rather than a limitation! Behind DDR there's human intervention, and human work establishes whether a given device has a given capability. Some capabilities can be detected algorithmically, but the returned value isn't always entirely reliable. Some other capabilities—the majority, in fact—can't be detected algorithmically but can be useful to know in order to intelligently decide which markup to serve to a given device.

Popular DDRs for the ASP.NET space are Wireless Universal Resource File, or WURFL, and 51degrees, as discussed in my previous column (msdn.microsoft.com/magazine/jj190798). Because both of these frameworks have a free offering for limited (mostly cloud-based) use, you might want to pick one of them even if you only need to determine whether a device is mobile or not. At least you'll save yourself the burden of frequently updating your IsMobile function to keep up with new devices and corner cases.

Optimize for Different Devices

Wrapping up, I believe that Web sites should offer an optimized set of pages to different devices, whether they're smartphones, tablets or laptops. This is best achieved if you can offer different Web sites or at least physically different views that you can manage and maintain separately. However, this separation should always be transparent to visitors of the site. The site should be reachable through a unique URL. You let the underlying platform do the magic of understanding the type of device and switch to the most appropriate view. In my next column I'll use DDRs to discuss an implementation of different views for different devices. ■

> The landing page will be a mobile-optimized page that shows a message to the user and offers links to the homepage of the desktop site or mobile site.

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: *Pranav Rastogi*

KENNY KERR

Lightweight Cooperative Multitasking with C++

If you work for a company that has one of those coding standards documents that would annihilate an entire rainforest were it ever to be printed, you'd better stop reading now. Chances are, what I'm about to show you will violate many of the sacred cows in the aforementioned epic. I'm going to tell you about a particular technique I originally developed to allow me to write completely asynchronous code efficiently and without the need for complex state machines.

Unless your name is Donald Knuth, it's likely any code you write will resemble something that has already been done. Indeed, the oldest account I can find of the technique I describe here is a mention by Knuth himself, and the most recent is by a gentleman from England by the name of Simon Tatham, known for the popular PuTTY terminal emulator. Nevertheless, to paraphrase the judge in the recent Oracle v. Google dispute, "you can't patent a *for* loop." Still, we're all indebted to our peers in the field of computer programming as we push the craft forward.

> If you were to ask C++ designer Bjarne Stroustrup what language he used to write C++, he would tell you it was C++.

Before I dive in and describe what my technique is and how it works, I need to present a quick diversion in the hope that it will give you a bit more perspective for what's to come. In "Compilers: Principles, Techniques and Tools, Second Edition" (Prentice Hall, 2006) by Alfred V. Aho, Monica S. Lam, Ravi Sethi and Jeffrey D. Ullman—more commonly known as the Dragon Book—the authors sum up the purpose of a compiler as being a program that can read a program in one language and translate it into an equivalent program in another language. If you were to ask C++ designer Bjarne Stroustrup what language he used to write C++, he would tell you it was C++. What this means is that he wrote a preprocessor that read C++ and produced C, which a standard C compiler could then further translate into some machine language. If you look close enough, you can see variants of this idea in many places. The C# compiler, for example, translates the seemingly magical *yield* keyword into a regular class that implements an iterator. Late last year, I noticed that the Visual C++ compiler first translated portions of the C++/CX syntax into standard C++ before compiling it further. This may have since changed, but the point is that compilers are fundamentally about translation.

Sometimes, when using a particular language, you might conclude that a feature would be desirable but the very nature of the language prohibits you from implementing it. This rarely happens in C++, because the language by design is suited to the expression of different techniques, thanks to its rich syntax and meta-programming facilities. However, this in fact did happen to me.

I spend most of my time these days working on an embedded OS that I created from scratch. Neither Linux, Windows nor any other OS runs under the covers. I rely on no open source software whatsoever, and indeed that would typically be impossible anyway for reasons that will become clear. This has opened my eyes to a whole world of C and C++ programming that's quite different from the traditional world of PC software development. Most embedded systems have very different constraints from those of "normal" programs. They have to be extremely reliable. Failure can be costly. Users are seldom around to "restart" a failed program. Systems might have to run for years uninterrupted and without human intervention. Imagine a world without Windows Update or the like. These systems might also have relatively scarce computing resources. Correctness, reliability and, in particular, concurrency all play central roles in the design of such systems.

As such, the plush world of Visual C++, with its powerful libraries, is seldom appropriate. Even if Visual C++ were to target my embedded microcontroller, the accompanying libraries aren't well-suited for systems with such scarce computing resources and, often, hard real-time constraints. One of the microprocessors I'm currently using has just 32KB of RAM running at less than 50 MHz, and this is still luxurious to some in the embedded community. It should be clear that by "embedded" I do not mean your average smartphone with a half-gig of RAM and a 1 GHz processor.

In "Programming: Principles and Practice Using C++" (Addison-Wesley Professional, 2008), Stroustrup cites free-store allocations (for example, *new* and *delete*), exceptions and dynamic_cast in a short list of features that must be avoided in most embedded systems because they aren't predictable. Unfortunately, that precludes the use of most of the standard, vendor-supplied and open source C++ libraries available today.

The result is that most embedded programming—and for that matter, kernel-mode programming on Windows—still employs C rather than C++. Given that C is primarily a subset of C++, I tend

to use a C++ compiler but stick to a strict subset of the language that's predictable, portable and works well in embedded systems.

This led me on a journey to find a suitable technique to enable concurrency in my little embedded OS. Up to this point, my OS had a single thread, if you can call it that. There are no blocking operations, so any time I needed to implement something that might take some time, such as waiting for a storage I/O interrupt or for a network retransmission timeout, I would need to write a carefully constructed state machine. This is standard practice with event-driven systems, but it results in code that's hard to reason through logically, because the code isn't sequential.

> What I'm saying is we should be able to write fast and responsive event-driven programs without involving a scheduler.

Imagine a storage driver. There might be a storage_read function to read a page of memory from a device's persistent flash storage. The storage_read function might first check whether the peripheral or bus is busy, and if so, simply queue the read request before returning. At some point the peripheral and bus become free and the request can proceed. This might involve disabling the transceiver, formatting a command appropriate for the bus, preparing the direct memory access buffers, enabling the transceiver again and then returning to allow the caller to do something else while the transfer completes in hardware. Eventually the bus signals its completion and the caller is notified via some callback mechanism, and any other queued requests proceed. Needless to say, it can get pretty complicated managing the queues, callbacks and state machines. Verifying everything is correct is even harder. I haven't even described how a nonblocking file system might be implemented on top of this abstraction or how a Web server might use the file system to serve up data—all without ever blocking. A different approach is needed to reduce the inevitable and growing complexity.

Now, imagine that C++ had a few keywords that let you transport the processor from one call stack to another in mid-function. Imagine the following hypothetical C++ code:

```
void storage_read(storage_args & args) async
{
  wait while (busy);

  busy = true;

  begin_transfer(args);

  yield until (done_transmitting());

  busy = false;
}
```

Notice the "async" contextual keyword after the parameter list. I'm also using two imaginary spaced keywords named "wait while" and "yield until." Consider what it means for C++ to have such keywords. The compiler would somehow have to express the notion of an interlude, if you will. Knuth called this a coroutine. The async keyword might appear with a function declaration to let the compiler know that the function can run asynchronously and must thus be called appropriately. The hypothetical wait and yield keywords are the actual points at which the function ceases to execute synchronously and can potentially return to the caller, only to resume where it left off at a later stage. You could also imagine a "wait until" conditional keyword as well as an unconditional yield statement.

I've seen alternatives to this cooperative form of concurrency—notably the Asynchronous Agents Library included with Visual C++—but all the solutions I found depended on some runtime scheduling engine. What I propose here and will illustrate in a moment is that it's entirely possible that a C++ compiler might indeed provide cooperative concurrency without any runtime cost whatsoever. Keep in mind that I'm not saying this will solve the manycore scalability challenge. What I'm saying is we should be able to write fast and responsive event-driven programs without involving a scheduler. And as with the existing C and C++ languages, nothing should prevent those techniques from being used along with OS threads or other concurrency runtimes.

Obviously, C++ doesn't support what I'm describing right now. What I discovered, however, is that it can be simulated reasonably well using Standard C or C++ and without relying on assembler trickery. Using this approach, the storage_read function described earlier might look as follows:

```
task_begin(storage_read, storage_args & args)
{
  task_wait_while(busy);

  busy = true;

  begin_transfer(args);

  task_yield_until(done_transmitting());

  busy = false;
}
task_end
```

Figure 1 **The Average Task**

```
struct average_args
{
  int * source;
  int sum;
  int count;
  int average;

  int task_;
};

task_begin(average, average_args & args)
{
  args.sum = 0;
  args.count = 0;
  args.average = 0;

  while (true)
  {
    args.sum += *args.source;
    ++args.count;
    args.average = args.sum / args.count;

    task_yield();
  }
}
task_end
```

Obviously, I'm relying on macros here. Gasp! Clearly, this violates item 16 in the C++ Coding Standards (bit.ly/8boiZ0), but the alternatives are worse. The ideal solution is for the language to support this directly. Alternatives include using longjmp, but I find that's worse and has its own pitfalls. Another approach might be to use assembly language, but then I'd lose all portability. It's debatable whether it could even be done as efficiently in assembly language, because that would most likely result in a solution that used more memory due to the loss of contextual information and the inevitable one-stack-per-task implementation. So humor me as I show you how simple and effective this is, and then how it all works.

To keep things clear, I'll henceforth call these asynchronous functions *tasks*. Given the task I described earlier, it can be scheduled simply by calling it as a function:

```
storage_args = { ... };
storage_read(args);
```

As soon as the task decides it can't proceed, it will simply return to the caller. Tasks employ a bool return type to indicate to callers whether they've completed. Thus, you could continuously schedule a task until it completes, as follows:

```
while (storage_read(args));
```

Of course, this would block the caller until the task completes. This might actually be appropriate, perhaps when your program first starts in order to load a configuration file or the like. Apart from that exception, you'd rarely want to block in this manner. What you need is a way to wait for a task in a cooperative manner:

```
task_wait_for(storage_read, args);
```

This assumes the caller is itself a task and will then yield to its caller until the nested task completes, at which point it will continue. Now let me loosely define the task keywords, or pseudo-functions, and then go through an example or two you can actually try for yourself:

- task_declare(name, parameter)
  Declares a task, typically in a header file.
- task_begin(name, parameter)
  Begins the definition of a task, typically in a C++ source file.
- task_end
  Ends the definition of a task.
- task_return()
  Terminates the execution of a task and returns control to the caller.
- task_wait_until(expression)
  Waits until the expression is true before continuing.
- task_wait_while(expression)
  Waits while the expression is true before continuing.
- task_wait_for(name, parameter)
  Executes the task and waits for it to complete before continuing.
- task_yield()
  Yields control unconditionally, continuing when the task is rescheduled.
- task_yield_until(expression)
  Yields control at least once, continuing when the expression is non-zero.

It's important to remember that none of these routines block in any way. They're all designed to achieve a high degree of concurrency in a cooperative manner. Let me illustrate with a simple example. This example uses two tasks, one to prompt the user for a number and the other to calculate a simple arithmetic mean of the numbers as they arrive. First is the average task, shown in **Figure 1**.

A task accepts exactly one argument that must be passed by reference and must include an integer member variable called task_. Obviously, this is something the compiler would hide from the caller given the ideal scenario of first-class language support. However, for the purpose of this simulation, I need a variable to keep track of the task's progress. All the caller needs to do is initialize it to zero.

The task itself is interesting in that it contains an infinite while loop with a task_yield call within the loop's body. The task initializes some state before entering this loop. It then updates its aggregates and yields, allowing other tasks to execute before repeating indefinitely.

Next is the input task, as shown in **Figure 2**.

This task is interesting in that it illustrates that tasks may in fact block, as the scanf_s function will do while it waits for input. Although not ideal for an event-driven system. This task also illustrates using the task_return call to complete the task in mid-function rather than using a conditional expression in the while statement. A task completes either by calling task_return or by falling off the end of the function, so to speak. Either way, the caller will see this as the task completing, and it will be able to resume.

To bring these tasks to life, all you need is a simple main function to act as a scheduler:

```
int main()
{
  int share = -1;

  average_args aa = { &share };
  input_args ia = { &share };

  while (input(ia))
  {
    average(aa);

    printf("sum=%d count=%d average=%d\n",
      aa.sum, aa.count, aa.average);
  }
}
```

The possibilities are endless. You could write tasks representing timers, producers and consumers, TCP connection handlers and much more.

Figure 2 **The Input Task**

```
struct input_args
{
  int * target;

  int task_;
};


task_begin(input, input_args & args)
{
  while (true)
  {
    printf("number: ");

    if (!scanf_s("%d", args.target))
    {
      task_return();
    }

    task_yield();
  }
}
task_end
```

MATCH YOUR SERVER TO YOUR BUSINESS.

ONLY PAY FOR WHAT YOU NEED!

With a 1&1 Dynamic Cloud Server, you can change your server configuration in real time.

- Independently configure CPU, RAM, and storage
- Control costs with pay-per-configuration and hourly billing
- Up to 6 Cores, 24 GB RAM, 800 GB storage
- 2000 GB of traffic included free
- Parallels® Plesk Panel 10 for unlimited domains, reseller ready
- Up to 99 virtual machines with different configurations

1&1 DYNAMIC CLOUD SERVER

3 MONTHS FREE!*

Base Configuration, Starting at $49.99/month

- **NEW:** Monitor and manage your cloud server through 1&1 mobile apps for Android™ and iPhone®.

1&1®

MEMBER OF united internet

www.1and1.com

*Offer valid for a limited time only. 3 months free only applies to $49.99 base configuration. Set-up fee of $49.00 applies. Base configuration includes 1 processor core, 1 GB RAM, 100 GB Storage. Other terms and conditions may apply. Visit www.1and1.com for full promotional offer details. Program and pricing specifications and availability subject to change without notice. 1&1 and the 1&1 logo are trademarks of 1&1 Internet, all other trademarks are the property of their respective owners.

So how does it work? First keep in mind again that the ideal solution is for the compiler to implement this, in which case it can use all kinds of clever tricks to implement it efficiently, and what I'm about to describe won't actually be anywhere near as sophisticated or complicated.

As best as I can tell, this comes down to a discovery by a programmer named Tom Duff, who found out that you can play clever tricks with the switch statement. As long as it's syntactically valid, you can nest various selection or iteration statements within a switch statement to effectively jump in and out of a function at will. Duff published a technique for manual loop unrolling, and Tatham then realized it could be used to simulate coroutines. I took those ideas and implemented tasks as follows.

The task_begin and task_end macros define the surrounding switch statement:

```
#define task_begin(name, parameter) \
                                    \
  bool name(parameter)              \
  {                                 \
    bool yield_ = false;            \
    switch (args.task_)             \
    {                               \
      case 0:

#define task_end                    \
                                    \
    }                               \
    args.task_ = 0;                 \
    return false;                   \
  }
```

It should now be obvious what the single task_ variable is for and how it all works. Initializing task_ to zero ensures that execution jumps to the beginning of the task. When a task ends, it's again set back to zero as a convenience so the task can be restarted easily. Given that, the task_wait_until macro provides the necessary jump location and cooperative return facility:

```
#define task_wait_until(expression)      \
                                         \
  args.task_ = __LINE__; case __LINE__:  \
  if (!(expression))                     \
  {                                      \
    return true;                         \
  }
```

The task_ variable is set to the predefined line number macro, and the case statement gets the same line number, thus ensuring that if the task yields, the next time it's scheduled the code will resume right where it left off. The remaining macros are shown in **Figure 3**.

> task_wait_until is also a great way to deal with out-of-memory conditions.

These should all be patently obvious. Perhaps the only subtlety worth mentioning is task_yield_until, as it's similar to task_wait_until but for the fact that it will always yield at least once. task_yield, in turn, will only ever yield exactly once, and I'm confident that any respectable compiler will optimize away my shorthand. I should mention that task_wait_until is also a great way to deal with out-of-memory conditions. Rather than failing in some deeply nested operation with dubious recoverability, you can simply yield until the memory allocation succeeds, giving other tasks an opportunity to complete and hopefully free up some much-needed memory. Again, this is critical for embedded systems where failure is not an option.

Figure 3 **The Remaining Macros**

```
#define task_return()                    \
                                         \
  args.task_ = 0;                        \
  return false;

#define task_wait_while(expression)      \
                                         \
  task_wait_until(!(expression))

#define task_wait_for(name, parameter)   \
                                         \
  task_wait_while(name(parameter))

#define task_yield_until(expression)     \
                                         \
  yield_ = true;                         \
  args.task_ = __LINE__; case __LINE__:  \
  if (yield_ || !(expression))           \
  {                                      \
    return true;                         \
  }

#define task_yield()                     \
                                         \
  task_yield_until(true)
```

Given that I'm emulating coroutines, there are some pitfalls. You can't reliably use local variables within tasks, and any code that violates the validity of the hidden switch statement is going to cause trouble. Still, given that I can define my own task_args—and considering how much simpler my code is thanks to this technique—I'm thankful that it works as well as it does.

I found it useful to disable the following Visual C++ compiler warnings:

```
#pragma warning(disable: 4127) // Conditional expression is constant
#pragma warning(disable: 4189) // Local variable is initialized but not referenced
#pragma warning(disable: 4706) // Assignment within conditional expression
```

Finally, if you're using the Visual C++ IDE, be sure to disable "edit and continue debugging" by using /Zi instead of /ZI.

As I concluded this column, I looked around the Web for any similar initiatives and found the new async and await keywords that the Visual Studio 2012 C# compiler has introduced. In many ways, this is an attempt to solve a similar problem. I expect the C++ community to follow suit. The question is whether these solutions will come to C and C++ in a manner that will produce predictable code suitable for embedded systems as I've described in this column or whether they'll rely on a concurrency runtime, as the current Visual C++ libraries do. My hope is that one day I'll be able to throw away these macros, but until that day comes, I'll remain productive with this lightweight, cooperative and multitasking technique.

Stay tuned for the next installment of Windows with C++ in which I'll show you some new techniques that Niklas Gustafsson and Artur Laksberg from the Visual C++ team have been working on to bring resumable functions to C++. ■

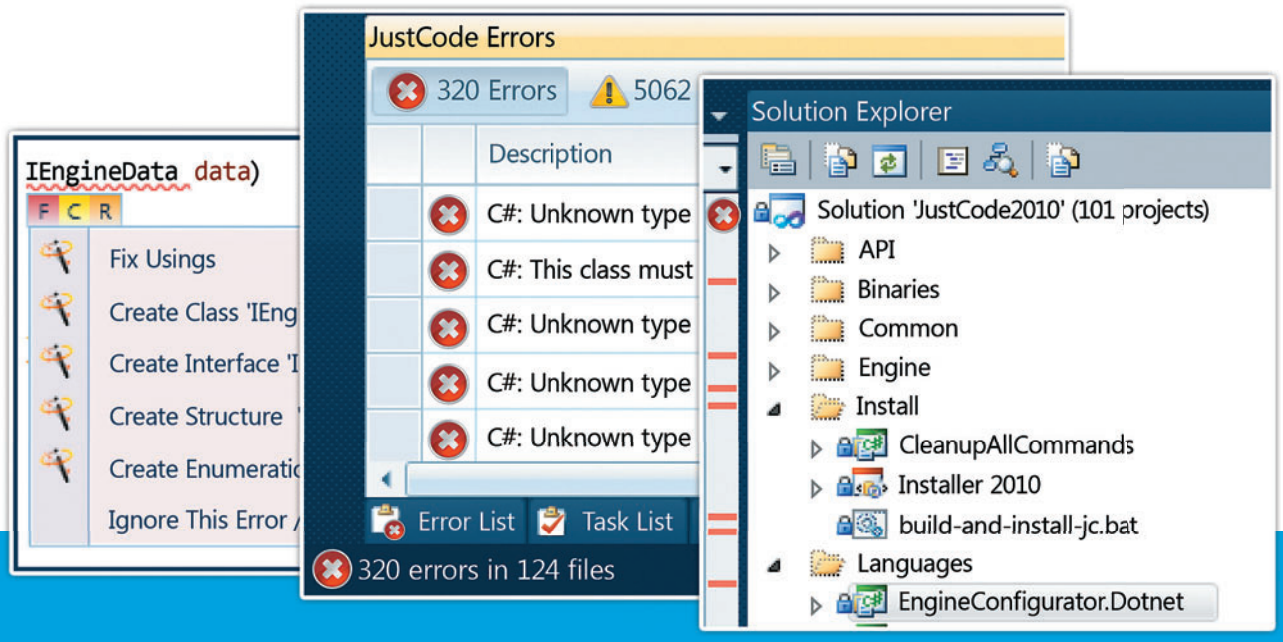
KENNY KERR *is a software craftsman with a passion for native Windows development. Reach him at kennykerr.ca.*

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

Code Faster, Code Smarter, JustCode

Your essential tool for all-out productivity and secret weapon for smart coding

NOW CLOUD-ENABLED



JustCode is a Visual Studio extension for solution-wide code analysis and error check, smart navigation and search. So you code faster. JustCode is unobtrusive and integrates seamlessly with your natural work-flow. Quick hints, code generation, the unit test runner and smart refactorings improve your code on the spot. So you code smarter.

Compatible with Visual Studio 2005, 2008, 2010 and 2012 (RC)

Download a free trial: www.telerik.com/justcode

2011 PARTNER OF THE YEAR
Mobility
Finalist

telerik
deliver more than expected

JULIE LERMAN

Pitfalls and Pointers for a Base Logging Class in EF Models

I recently spent time with a client who was experiencing some occasional—but severe—performance issues with their Entity Framework-related code. Using a tool to profile the queries generated by Entity Framework, we discovered a 5,800-line SQL query hitting the database. (Learn about profiling tools in my December 2010 Data Points column, "Profiling Database Activity in the Entity Framework," at msdn.microsoft.com/magazine/gg490349.) I gasped when I saw that the EDMX model contained an inheritance hierarchy I had taught friends, loved ones and developers to avoid. The model had a single base entity from which every other entity derived. The base entity was used to ensure each entity had properties to track logging data such as DateCreated and DateLastModified. Because this model was created using Model First, the inheritance was interpreted by Entity Framework as a Table per Type (TPT) model in which each entity mapped to its own table in the database. To the uninitiated, this looks innocent enough.

But TPT inheritance is notorious in Entity Framework for its generic query-building pattern that can result in sprawling, poorly performing SQL queries. You might be starting out with a new model where you can avoid TPT, or you might already be stuck with an existing model and TPT inheritance. Either way, this month's column is dedicated to helping you comprehend the potential performance pitfalls of TPT in this scenario, and showing you a few tricks you can leverage to get around them if it's too late to modify the model and database schema.

The Scary Model

Figure 1 shows an example of a model I've seen all too often. Notice the entity named TheBaseType. Every other entity derives from it in order to automatically inherit a DateCreated property. I understand why it's tempting to design it this way, but Entity Framework schema rules also demand that the base type owns the key property of each derived entity. To me, that's already a red flag signaling it's not an appropriate use of inheritance.

It's not that Entity Framework specifically created a problem with this design; it's a design flaw in the model itself. In this case, inheritance says that Customer is a TheBaseType. What if we changed the name of that base entity to "LoggingInfo" and then repeated the statement "Customer is a LoggingInfo"? The fallacy of the statement becomes more obvious with the new class name. Compare that to Customer is a Person. Perhaps I've now convinced you to avoid doing this in your models. But if not, or if you're already stuck with this model, let's take it a little further.

By default, the Model First workflow defines a database schema with one-to-one relationships between the base table and all of the tables that represent the derived types. This is the TPT hierarchy mentioned earlier.

If you were to execute a few simple queries, you might not notice any problem—especially if, like me, you're not a DBA or other flavor of database guru.

For example, this LINQ to Entities query retrieves the DateCreated property for a particular customer:

```
context.TheBaseTypes.OfType<Customer>()
  .Where(b => b.Id == 3)
  .Select(c => c.DateCreated)
  .FirstOrDefault();
```

The query results in the following TSQL executed in the database:

```
SELECT TOP (1)
[Extent1].[DateCreated] AS [DateCreated]
FROM  [dbo].[TheBaseTypes] AS [Extent1]
INNER JOIN [dbo].[TheBaseTypes_Customer] AS [Extent2] ON [Extent1].[Id]
= [Extent2].[Id]
WHERE 3 = [Extent1].[Id]
```

It's a perfectly good query.

A query to retrieve an entire entity is a little uglier because of the need to perform a nested query. The base query retrieves all of the fields that represent the join between TheBaseTypes table and the table containing the derived type. Then a query over those results

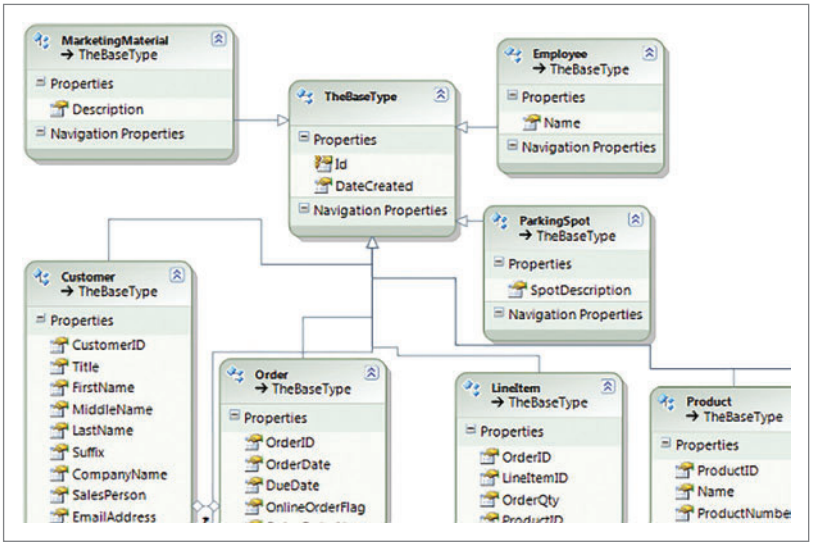


Figure 1 **All Classes Inherit from TheBaseType**

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

projects the fields to be returned to Entity Framework to populate the type. For example, here's a query to retrieve a single product:

```
context.TheBaseTypes.OfType<Product>().FirstOrDefault();
```

Figure 2 shows the TSQL executed on the server.

That's still not such a bad query. If you were profiling your queries up to this point, you might not notice any issues caused by the model's design.

But what about this next "simple" query, which wants to find all objects that were created today, regardless of type? The query could return Customers, Products, Orders, Employees or any other type in your model that derives from the base. Thanks to the model design, this seems like a reasonable request, and the model plus LINQ to Entities makes it easy to express (DateCreated is stored in the database as a date type, so I don't have to be concerned about comparing to DateTime fields in my example queries):

```
var today= DateTime.Now.Date;
context.TheBaseTypes
  .Where(b => b.DateCreated == today)
  .ToList();
```

Expressing this query in LINQ to Entities is short and sweet. But don't be fooled. It's a hefty request. You're asking EF and your database to return instances of any type (be it Customer or Product or Employee) created today. Entity Framework must begin by querying each table that maps to the derived entities and joining each one to the single related TheBaseTypes table with the DateCreated field. In my environment, this creates a 3,200-line query (when that query is nicely formatted by EFProfiler), which can take Entity Framework some time to build and the database some time to execute.

In my experience, a query like this belongs in a business analysis tool anyway. But what if you've got the model and you want to get that info from within your app, perhaps for an administrative reporting area of an application you're building? I've seen developers try to do this type of query in their applications, and I still say you need to think outside of the Entity Framework box. Build the logic into the database as a view or stored procedure and call that from Entity Framework, rather than asking EF to build the query for you. Even as a database procedure, this particular logic isn't simple to build. But there are benefits. First, you have a greater chance of building a better-performing query. Second, EF won't have to take the time to figure out the query. Third, your application won't have to send a 3,300 (or more!)-line query across the pipe. But be warned that the more you dig into this problem and attempt to solve it from within the database or by using EF and .NET coding logic, the clearer it will become that the problem is not so much Entity Framework as the overall model design that's getting in your way.

If you can avoid querying from the base type and query-specific types, your queries will be much simpler. Here's an example that expresses the previous query to focus on a particular type:

```
context.TheBaseTypes.TypeOf<Product>()
  .Where(b => b.DateCreated == today)
  .ToList();
```

Because EF didn't have to be prepared for every type in the model, the resulting TSQL is a simple 25-line query. With the DbContext API, you don't even have to use TypeOf to query derived types. It's possible to create DbSet properties for the derived types. So I could query even more simply:

```
context.Products
  .Where(b => b.DateCreated == today)
  .ToList();
```

In fact, for this model I'd completely remove the TheBaseTypes DbSet from my context class and prevent anyone from expressing queries directly from this base type.

Figure 2 **Partial Listing of a Nested TSQL Query When Specifying a Type**

```
SELECT
[Limit1].[Id] AS [Id],
[Limit1].[C1] AS [C1],
[Limit1].[DateCreated] AS [DateCreated],
[Limit1].[ProductID] AS [ProductID],
[Limit1].[Name] AS [Name],
[...continued list of fields required for Product class...]
FROM ( SELECT TOP (1)
        [Extent1].[Id] AS [Id],
        [Extent1].[DateCreated] AS [DateCreated],
        [Extent2].[ProductID] AS [ProductID],
        [Extent2].[Name] AS [Name],
        [Extent2].[ProductNumber] AS [ProductNumber],
        [...continued list of fields from Products table aka "Extent2" ...],
        [Extent2].[ProductPhoto_Id] AS [ProductPhoto_Id],
        'OXOX' AS [C1]
        FROM  [dbo].[TheBaseTypes] AS [Extent1]
        INNER JOIN [dbo].[TheBaseTypes_Product] AS [Extent2] ON [Extent1].
[Id] = [Extent2].[Id]
)  AS [Limit1]
```

Logging Without the Scary Model

I've focused so far on a hierarchy scenario I strongly advise developers to avoid when building models with Entity Framework: using a mapped entity as a base from which every single entity in the model also derives. Sometimes I come upon scenarios where it's just too late to change the model. But other times I'm able to help my clients avoid this path completely (or they're early enough in development that we're able to change the model).

So how to better achieve the goal—which is to provide commonly tracked data, such as logging data—for every type in your model?

Often, the first thought is to keep the inheritance but change the type of hierarchy. With Model First, TPT is the default but you can change that to Table per Hierarchy (TPH) using the Entity Designer Generation Power Pack (available in Visual Studio Gallery via Extension Manager). Code First defaults to TPH when you define inheritance in your classes. But you'll quickly see that this is not a solution at all. Why? TPH means that the entire hierarchy is contained in a single table. In other words, your database would consist of just one table. I'm hoping no more explanation is necessary to convince you that this is not a good path.

As I said earlier (when I asked if a Customer is really a type of LoggingInfo), the specific scenario I've focused on, to solve the problem of tracking common data, begs that you just avoid inheritance altogether for that goal. I'd recommend you consider an interface or complex types instead, which will embed the fields into each table. If you're already stuck with the database that created a separate table, a relationship will do.

To demonstrate, I'll switch to a model based on classes using Code First instead of an EDMX (although you can achieve these same patterns using an EDMX model and the designer).

In the first case I'll use an interface:

```
public interface ITheBaseType
{
  DateTime DateCreated { get; set; }
}
```

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

Each class will implement the interface. It will have its own key property and contain a DateCreated property. For example, here's the Product class:

```
public class Product : ITheBaseType
{
  public int ProductID { get; set; }
  // ...other properties...
  public DateTime DateCreated { get; set; }
}
```

In the database, each table has its own DateCreated property. Therefore, repeating the earlier query against the Products creates a straightforward query:

```
 context.Products
.Where(b => b.DateCreated == today)
.ToList();
```

Because all of the fields are contained in this table, I no longer need a nested query:

```
SELECT TOP (1) [Extent1].[Id]                    AS [Id],
               [Extent1].[ProductID]             AS [ProductID],
               [Extent1].[Name]                  AS [Name],
               [Extent1].[ProductNumber]         AS [ProductNumber],
               ...more fields from Products table...
               [Extent1].[ProductPhoto_Id]       AS [ProductPhoto_Id],
               [Extent1].[DateCreated]           AS [DateCreated]
FROM   [dbo].[Products] AS [Extent1]
WHERE  [Extent1].[DateCreated] = '2012-05-25T00:00:00.00'
```

If you prefer to define a complex type and reuse that in each of the classes, your types might look like this:

```
public class Logging
{
  public DateTime DateCreated { get; set; }
}

public class Product{
  public int ProductID { get; set; }
  // ...other properties...
  public Logging Logging { get; set; }
}
```

Note that the Logging class doesn't have a key field (such as Id or LoggingId). Code First conventions will presume this to be a complex type and treat it as such when it's used to define properties in other classes, as I've done with Product.

The Products table in the database has a column generated by Code First called Logging_DateCreated, and the Product.Logging.DateCreated property maps to that column. Adding the Logging property to the Customer class would have the same effect. The Customers table will also have its own Logging_DateCreated property, and it maps back to Customer.Logging.DateCreated.

In code, you'll need to navigate through the logging property to reference that DateCreated field. Here's the same query as before, rewritten to work with the new types:

```
context.Products.Where(b => b.Logging.DateCreated == DateTime.Now).ToList();
```

The resulting SQL is the same as the interface example except the field name is now Logging_DateCreated rather than DateCreated. It's a short query that only queries the Products table.

One of the benefits of inheriting from the class in the original model is that it's easy to code logic to automatically populate the fields from the base class—during SaveChanges, for example. But you can create logic just as easily for the complex type or for the interface, so I don't see any disadvantage with these new patterns. **Figure 3** shows a simple example of setting the DateCreated property for new entities during SaveChanges (you can learn more about this technique in the Second and DbContext editions of my "Programming Entity Framework" book series).

Figure 3 **Setting the Interface's DateCreated Property During SaveChanges**

```
public override int SaveChanges()
{
  foreach (var entity in this.ChangeTracker.Entries()
    .Where(e =>
    e.State == EntityState.Added))
  {
    ApplyLoggingData(entity);
  }
  return base.SaveChanges();
}

private static void ApplyLoggingData(DbEntityEntry entityEntry)
{
  var logger = entityEntry.Entity as ITheBaseType;
  if (logger == null) return;
  logger.DateCreated = System.DateTime.Now;
}
```

Some Changes in EF 5

Entity Framework 5 does bring some improvements to queries that are generated from TPT hierarchies that help but do not alleviate the problems I demonstrated earlier. For example, rerunning my query that initially resulted in 3,300 lines of SQL on a machine that has the Microsoft .NET Framework 4.5 installed (with no other changes to the solution) generates a query that's reduced to 2,100 lines of SQL. One of the biggest differences is that EF 5 doesn't rely on UNIONs to build the query. I'm not a DBA, but my understanding is that such an improvement wouldn't impact the performance of the query in the database. You can read more about this change to TPT queries in my blog post, "Entity Framework June 2011 CTP: TPT Inheritance Query Improvements," at bit.ly/MDSQuB.

Not All Inheritance Is Evil

Having a single base type for all entities in your model is an extreme example of modeling and inheritance gone wrong. There are many good cases for having an inheritance hierarchy in your model—for example, when you do want to describe that a Customer is a Person. What's also important is the lesson that LINQ to Entities is just one tool that's available to you. In the scenario my client showed me, a clever database developer reconstructed the query against the base type fields as a database stored procedure, which brought a multisecond activity down to one that took just 9 milliseconds. And we all cheered. We're still hoping they'll be able to redesign the model and tweak the database for the next version of the software, though. In the meantime, they're able to let Entity Framework continue to generate those queries that aren't problematic and use the tools I left behind to tweak the application and database for some fantastic performance improvements. ■

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

THANKS to the following technical expert for reviewing this article: *Diego Vega*

Raising the Bar...

And Pie, and Spline and Scatter... on Mobile Business Intelligence

Compatible with
Microsoft® Visual Studio® 2012

Visualize your BI in 50+ ways, and on just as many devices. From candlesticks to polar and radial charts, nobody offers the breadth and depth of dynamic, high fidelity, totally mobile charting solutions that you get from Infragistics' NetAdvantage. **Try a free, fully supported trial of NetAdvantage for .NET today!**

www.infragistics.com/NET

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

HTML 5

JOSEPH FULTZ AND CHRIS MABRY

Decoupling the Cloud with MEF

A colleague and I have been working on a project over the past several months that leverages the Microsoft Extensibility Framework (MEF). In this article, we'll look at how you might use MEF to make a cloud deployment a little more manageable and flexible. MEF—and similar frameworks such as Unity—are the software fabric that frees developers from managing dependency resolution, object creation and instantiation. Now and again you might find yourself writing a factory method or creating dependent objects inside of a constructor or required initialization method, but for the most part such work is no longer necessary thanks to frameworks such as MEF.

By using MEF in our deployment in conjunction with the StorageClient API, we can deploy and make available new classes without recycling or redeploying our Web roles. Moreover, we can deploy updated versions of types into the cloud without a full redeploy and simply recycle the application instead. Note that while we're using MEF here, following a similar structure using Unity, Castle Windsor, StructureMap or any of the other similar containers should net the same results, with the primary differences being syntax and type registration semantics.

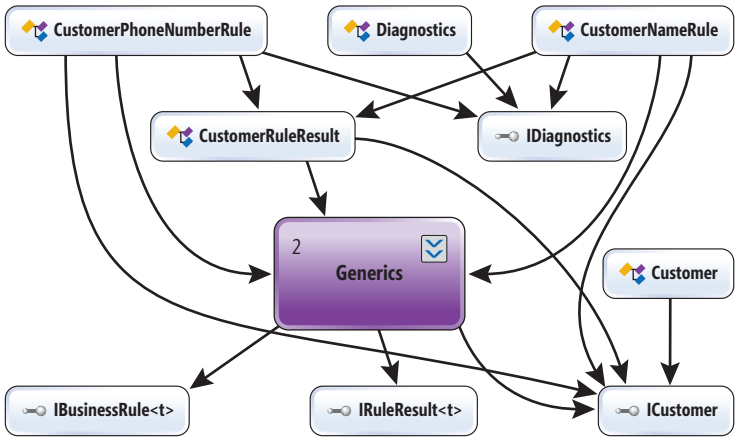


Figure 1 **Class Diagram**

Design and Deployment

As the saying goes: To get a little more out, you have to put a little more in. In this case that requires certain construction standards and some additional work around the deployment. First, if you're used to using a dependency injection (DI) or composition container, chances are you're pretty keen on keeping implementation and interface separated within your code. We don't stray from that goal here—all our concrete class implementations have inheritance that traces back to an interface type. That doesn't mean every class will directly inherit from an interface, but classes will generally have layers of abstraction that follow a pattern like Interface → Virtual → Concrete.

Figure 1 shows that not only does the primary class I'm interested in have such a chain, but in fact one of its required properties is also abstracted. All of the abstraction makes it easier to replace parts or add additional functionality in the form of a new library that exports the desired contract (in this case the interface). Beyond composition, a nice side effect of being a martinet about abstracting your class design is that it better enables testing via mocked interfaces.

The harder part of the requirement is the change in the deployment model for the application. Because we want to build our catalog of imports and exports at run time, and refresh it without having to deploy again, we have to deploy the binaries that hold our concrete classes outside of the Web role deployment. That also forces a little extra work for the application at startup. **Figure 2** depicts the startup work in the Global.asax as it calls into a helper class that we've created named MEFContext.

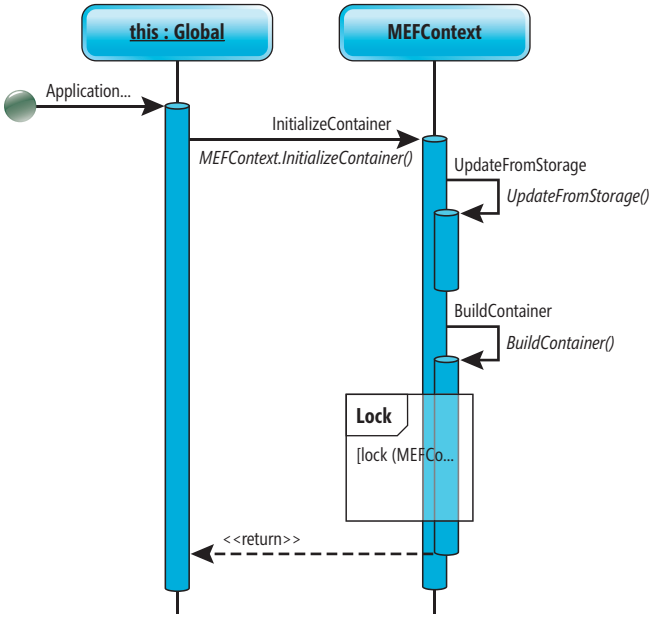
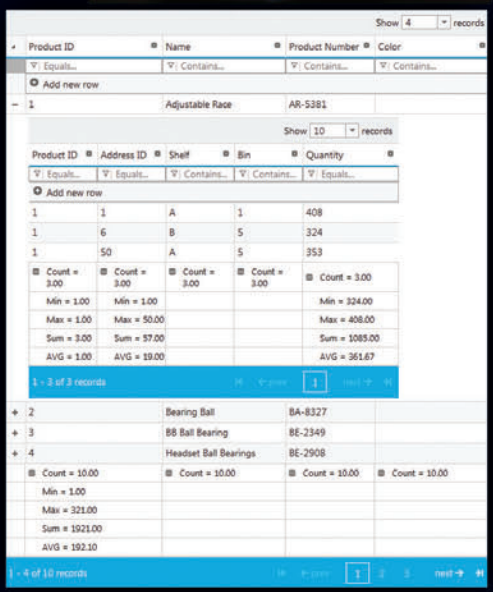


Figure 2 **Building the Catalog at Startup**

MetroTactual

[me-troh tak-choo-uhl]



Compatible with
Microsoft® Visual Studio® 2012

noun, adjective

1. Modern, clean, sleek, stylish, touch-friendly design and UX
2. Feng Shui for your apps
3. Available in NetAdvantage 12.1 toolsets

See also: NetAdvantage for .NET

Try your free, fully supported trial today.
www.infragistics.com/NET

INFRAGISTICS™
DESIGN / DEVELOP / EXPERIENCE

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

Figure 3 **First Half of UpdateFromStorage**

```
// Could also pull from config, etc.
string containerName = CONTAINER_NAME;

// Using development storage account
CloudStorageAccount storageAccount = CloudStorageAccount.DevelopmentStorageAccount;

// Create the blob client and use it to create the container object
CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();

// Note that here is where the container name is passed
// in order to get to the files we want
CloudBlobContainer blobContainer = new CloudBlobContainer(
  storageAccount.BlobEndpoint.ToString() + "/" + containerName,
  blobClient);

// Create the options needed to get the blob list
BlobRequestOptions options = new BlobRequestOptions();
options.AccessCondition = AccessCondition.None;
options.BlobListingDetails = BlobListingDetails.All;
options.UseFlatBlobListing = true;
options.Timeout = new TimeSpan(0, 1, 0);
```

Figure 4 **Second Half of UpdateFromStorage**

```
// Iterate over the collect
// Grab the files and save them locally
foreach (IListBlobItem item in blobs)
{
  string fileAbsPath = item.Uri.AbsolutePath.ToLower();
  // Just want the file name ...
  fileAbsPath = fileAbsPath.Substring(fileAbsPath.LastIndexOf('/') + 1);

  try
  {
    Microsoft.WindowsAzure.StorageClient.CloudPageBlob pageblob =
      new CloudPageBlob(item.Uri.ToString());
    pageblob.DownloadToFile(MEFContext.CacheFolderPath + fileAbsPath, options);
  }
  catch (Exception)
  {
    // Ignore exceptions, if we can't write it's because
    // we've already got the file, move on
    }
}
```

Runtime Composition

Because we're going to be loading the catalog from files in storage, we'll have to get those files into our cloud storage container. Therefore, getting the files into the Windows Azure Storage location needs to become part of the deployment process. This is probably most easily done using Windows Azure PowerShell cmdlets (wappowershell.codeplex.com) and some post-build steps. For our purposes, we'll manually move the binaries using the Windows Azure Storage Explorer (azurestorageexplorer.codeplex.com).

We created a project that contains a common diagnostics class, a customer entity and a couple of rule libraries. All of the rule libraries have to inherit from and export an interface of type IBusinessRule<t>, where t represents the entity against which rules are enforced. Here are the import parts of the class declaration for a rule:

```
[Export(typeof(IBusinessRule<ICustomer>))]
public class CustomerNameRule : IBusinessRule<ICustomer>
{
  [Import(typeof(IDiagnostics))]
  IDiagnostics _diagnostics;
    ...
}
```

You can see the export as well as the diagnostics dependency that MEF will inject for us when we ask for the rule object. It's important to know what's being exported as that will in turn be the contract by which you resolve the instances you want. The Microsoft .NET Framework 4.5 will bring some enhancements to MEF that will allow a loosening of some of the constraints currently around generics in the container. For example, currently you can register and retrieve something such as IBusinessRule<ICustomer>, but not something like IBusinessRule<t>. Sometimes you want all instances of a type beyond its actual template type. Currently, the easiest way to accomplish this is to register a string contract name that's an agreed convention in your project or solution. For our sample, a declaration like the preceding will work.

We have two rules, one for phone number and one for name, and a diagnostics library, each of which will be available through the MEF container. The first thing we have to do is to grab the libraries out of Windows Azure Storage and bring them down to a local resource (local directory) so we can load them with a DirectoryCatalog. To do this, we include a couple of function calls in the Application_Start of the Global.asax:

```
// Store the local directory for later use (directory catalog)
MEFContext.CacheFolderPath = RoleEnvironment.GetLocalResource("ResourceC
ache").RootPath.ToLower();
MEFContext.InitializeContainer();
```

We're just grabbing the needed resource path, which is configured as part of the Web role, and then calling the method to set up the container. That initialization method in turn calls UpdateFromStorage to get the files and BuildContainer to create the catalog and then the MEF container.

The UpdateFromStorage method looks in a predetermined container and iterates over the files in the container, downloading each of them into the local resource folder. The first part of this method is shown in **Figure 3**.

In the first half we set up the storage client to fetch what we need. For this scenario, we're asking for whatever is there. In cases where

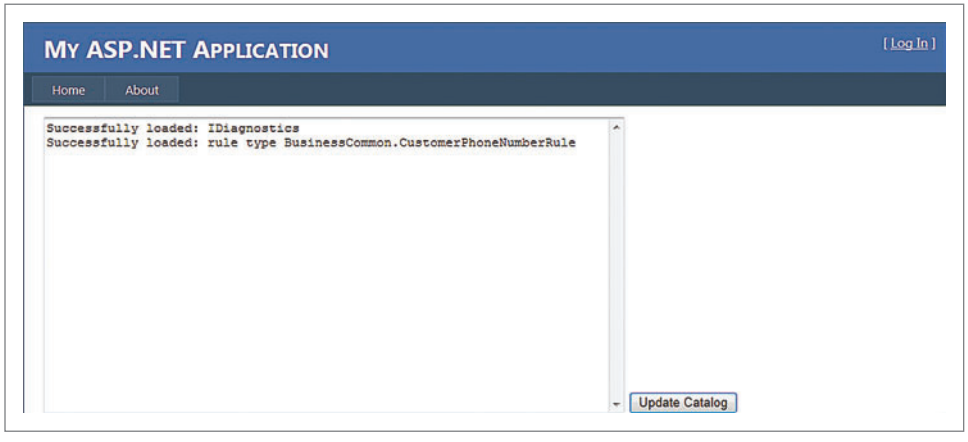


Figure 5 **Initial Exports**

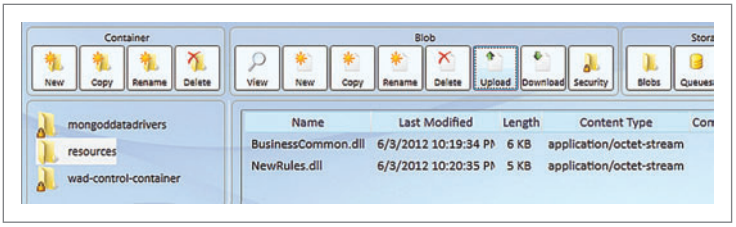


Figure 6 **New Binary in Windows Azure Storage Explorer**

you're bringing files down from storage to a local resource, it might be worth doing a full pass and getting everything. For a more targeted fetch of the files, you could assign some IfMatch condition to the options.AccessCondition property. This would require that etags be set on the blobs when uploaded. Additionally, you could optimize the update side of rebuilding the MEF container by storing the last update time and applying an AccessCondition of IfModifiedSince.

> If you're used to using a dependency injection (DI) or composition container, chances are you're pretty keen on keeping implementation and interface separated within your code.

Figure 4 shows the second half of UpdateFromStorage.

Once the storage client is ready, we simply iterate over the blob items and download them to the resource. Depending on the conditions and goals of the overall download, you could replicate folder structures locally in this operation or build a folder structure based on convention. Sometimes a folder structure is a requirement to avoid name collisions. We're just going with the shotgun method and grabbing all of the files and sticking them in one place because we know it's just two or three DLLs for this sample.

With this, we have the files in place and just need to build the container. In MEF, the composition container is built from one or more catalogs. In this case, we're going to use a DirectoryCatalog because this makes it easy to simply point the catalog to the directory and load the binaries that are available. Thus, the code to register the types and prepare the container is short and simple:

```
// Store the container for later use (resolve type instances)
var catalog = new DirectoryCatalog(CacheFolderPath);
MEFContainer = new CompositionContainer(catalog);
MEFContainer.ComposeParts();
```

Now we'll run the site and we should see a dump of the types available in the container, as shown in **Figure 5**.

We're not dumping the entire container here, but rather asking specifically for the IDiagnostics interface and then all exports of type IBusinessRule<ICustomer>. As you can see, we have one of each of these prior to uploading a new business rule library into the storage container.

Adding New Functionality at Run Time

Due to the way the AppDomain handles type loading and resolution, we can't just refresh a type at run time after it has been loaded once. However, we can add functionality by placing a new binary into our directory and reloading the DirectoryCatalog, as shown in **Figure 6**.

The BusinessCommon.dll contains our initial rule and the diagnostics library, but we've decided we need to add another rule. Like the phone number rule, this one checks for null or empty values for either of the name fields on the Customer entity. If the rule is violated, this is added to the rule results and logs using an imported IDiagnostics interface (see **Figure 7**).

We've placed NewRules.dll into the storage location and now need to get it loaded into the application. Ideally, you want to trigger the container rebuild by doing a little bit of file watching on the storage container. Again, this is easily accomplished with a quick poll using the IfModifiedSince AccessCondition. However, we've opted for the more manual process of clicking Update Catalog on our test app. **Figure 8** shows the results.

We just repeat the steps to create the catalog and initialize the container, and now we have a new rule library to enforce. Note that we haven't restarted the app or redeployed, but we have new code running in the environment. The only loose end here is that some synchronization method is needed, because we can't have code trying to use the composition container while we're replacing the reference:

```
var catalog = new DirectoryCatalog(CacheFolderPath);
CompositionContainer newContainer = new CompositionContainer(catalog);
newContainer.ComposeParts();
lock(MEFContainer)
{
  MEFContainer = newContainer;
}
```

The primary reason for building a secondary container and then just replacing the reference is to reduce the lock quantum and return the container to use right away.

To further evolve the code base, the next step would be to implement your own custom catalog type—for example, AzureStorageCatalog,

Figure 7 **Name Rule**

```
[Export(typeof(IBusinessRule<ICustomer>))]
public class CustomerNameRule : IBusinessRule<ICustomer>
{
  [Import(typeof(IDiagnostics))]
  IDiagnostics _diagnostics;

  public void ProsecuteRules(ref ICustomer Entity,
    ref List<IRuleResult<ICustomer>> RulesResults)
  {
    if (string.IsNullOrEmpty(Entity.FirstName) ||
      string.IsNullOrEmpty(Entity.LastName))
    {
      // Rule violation, add to collection
      CustomerRuleResult result = new CustomerRuleResult()
      {
        TestedEntity = Entity,
        RuleMessage = "The phone first and last names cannot be null or empty.",
        RuleName = "Full Name Required"
      };
      // Log execution result
      _diagnostics.Log("Name rule violation");

      RulesResults.Add(result);

    }
  }
```

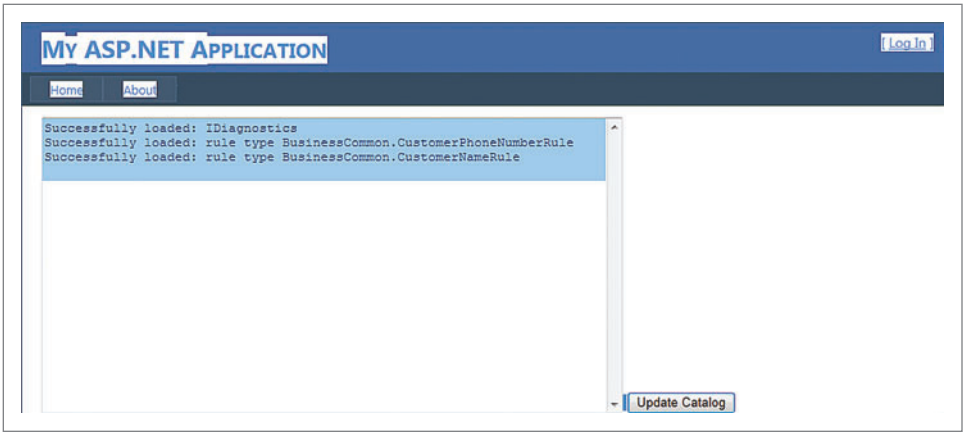


Figure 8 **Updated Rules Exports**

as shown in **Figure 9**. Unfortunately, the current object model doesn't have a proper interface or an easily reusable base defined, so using a bit of inheritance as well as some encapsulation is probably the best bet. Implementing a class similar to the AzureStorageCatalog listing would enable a simple model of instantiating the custom catalog and using it directly in the composition container.

Updating Existing Functionality

Adding new functionality to our deployment was pretty easy, but we don't have the same good news for updating existing functionality or libraries. Though the process is better than a complete redeployment, it's still fairly involved because we have to move the files to storage and the relevant Web roles have to update their local resource folders. However, we'll also recycle the roles because we need to unload and reload the AppDomain to refresh the type definition stored in the container. Even if you load the Composition Container and types into a secondary AppDomain and try to load from there, the AppDomain in which you're requesting the type will load it from previously loaded metadata. The only way around this we could see would be to send the entities to the secondary AppDomain and add some custom marshaling rather than using the exported types on the primary AppDomain. That pattern seems problematic to us; the double AppDomain in itself seems problematic. Thus, a simpler solution is to recycle the roles after the new binaries are made available.

There's some good news regarding Windows Azure update domains. Take a look at my February 2012 column, "Windows Azure Deployment Domains" (msdn.microsoft.com/magazine/hh781019), which describes walking the update domains and restarting instances in each. On the positive side, the site stays up with no need for a full redeployment. However, you could potentially experience two different behaviors during the refresh. This is an acceptable risk, though, because the same would be true during a rolling update if you did a full deployment.

Figure 9 **AzureStorageCatalog**

```
public class AzureStorageCatalog:ComposablePartCatalog
{
  private string _localCatalogDirectory = default(string);
  private DirectoryCatalog _directoryCatalog = default(DirectoryCatalog);

  AzureStorageCatalog(string StorageSetting, string ContainerName)
    :base()
  {
    // Pull the files to the local directory
    _localCatalogDirectory = GetStorageCatalog(StorageSetting, ContainerName);
    // Load the exports using an encapsulated DirectoryCatalog
    _directoryCatalog = new DirectoryCatalog(_localCatalogDirectory);

  }

  // Return encapsulated parts
  public override IQueryable<ComposablePartDefinition> Parts
  {
    get { return _directoryCatalog.Parts; }
  }

  private string GetStorageCatalog(string StorageSetting, string ContainerName)
  {  }
}
```

> It's important to know what's being exported as that will in turn be the contract by which you resolve the instances you want.

You could configure this to happen within the deployment, but the problem is one of coordination. To do this would require that the restarts of the instances be coordinated, so the instances would either need to elect a leader or have some voting system. Rather than writing some artificial intelligence into the Web roles, we feel the task is more easily handled by a monitoring process and the Windows Azure cmdlets referenced earlier.

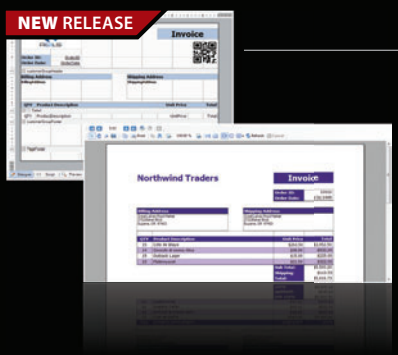
There are many reasons to use a framework such as MEF that are beyond the narrow bit of functionality we've highlighted here. What we wanted to highlight is that, by using the inherent capabilities of Windows Azure in combination with a composition/DI/Inversion of Control-type framework, you could create a dynamic cloud application that could easily respond to the last-minute changes that always seem to pop up. ■

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.*

Chris Mabry *is a lead developer at Hewlett-Packard Co. with a current focus on leading a team to deliver a rich UI experience based on service-enabled client frameworks.*

Thanks to the following technical expert for reviewing this article: *Chris Brooks*

NEW RELEASE



ActiveReports Developer 7 | from $685.02

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

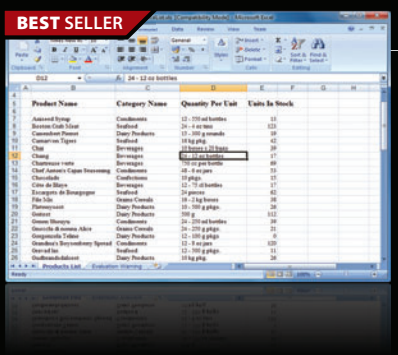
GdPicture.NET | from $3,919.47

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

BEST SELLER



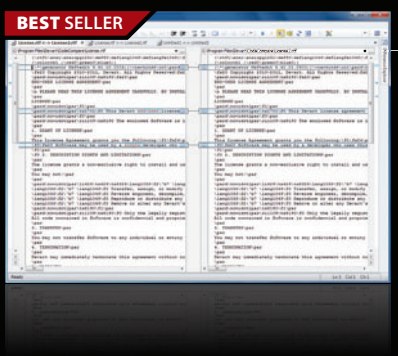
Aspose.Total for .NET | from $2,449.02

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



CodeCompare Pro | from $48.95

devart

An advanced visual file comparison tool with Visual Studio integration.

- Replaces the default diff / merge tools in SVN, TFS, Git, Mercurial or any other VCS
- Visual Studio integration for development and merging operations within your IDE
- Three-way file merging allows you to merge two derived files to a base / result file
- Folder Synchronization lets you compare and merge entire folders
- Code Orientation offers syntax highlighting plus structure and lexical comparison algorithms

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

Functional-Style Programming in C++

David Cravey

C++ is a multiparadigm, systems-level language that provides high-level abstractions with very low (often zero) runtime cost. The paradigms commonly associated with C++ include procedural, object-oriented and generic programming. Because C++ provides excellent tools for high-level programming, even functional-style programming is quite reasonable.

By functional-style programming, I don't mean the programming is strictly functional, just that it's easy to use many of the functional building blocks in C++. This article will focus on one of the most important functional programming constructs: working with values as opposed to identities. I'll discuss the strong support C++ has always had for working with values, then show how the new C++ 11 standard expands this support with lambdas. Finally, I'll introduce a method of working with immutable data structures that maintains the speed C++ is known for while providing the protection that functional languages have long enjoyed.

Values vs. Identities

Let me first explain what I mean by working with values rather than identities. Simple values such as 1, 2 and 3 are easy to identify. I could also say that 1, 2 and 3 are constant integer values. This would be redundant, however, because all values are actually constants and the values themselves never change (that is, 1 is always 1 and 1 will never be 2). On the other hand, the value associated with an identity may change (*x* might equal 1 now, but it could equal 2 later).

This article discusses:

- Values versus identities
- Lambdas
- Immutable data types

Technologies discussed:

C++ 11

Code download available at:

archive.msdn.microsoft.com/mag201208CPP

Figure 1 **Using a Value Type**

```
void Foo()
{
  for (int x = 0; x < 10; ++x)
  {
    // Call Bar, passing the value of x and not the identity
    Bar(x);
  }
}

void Bar(int y)
{
  // Display the value of y
  cout << y << " ";

  // Change the value that the identity y refers to
  // Note: This will not affect the value that the variable x refers to
  y = y + 1;
}

// Outputs:
// 0 1 2 3 4 5 6 7 8 9
```

Unfortunately, it's easy to confuse values and value types. Value types are passed around by value rather than by reference. Though I want to focus here on the values and not the mechanism involved in using or copying them, it's useful to see how value types go part way in preserving the concept of values versus identities.

The code in **Figure 1** demonstrates a simple use of a value type.

With only a small change, the variable *y* can become a reference type—which drastically changes the relationship between *x* and *y*, as shown in **Figure 2**.

As **Figure 3** shows, C++ also provides the const modifier, which prevents the programmer from making changes to a variable and thus further preserves the concept of a value. (As with most things in C++, however, there's at least one way to defeat that protection. For more information, look up const_cast, which is intended for working with older code that isn't "const correct.")

Note in **Figure 3** that though *y* is passed by reference, the value of *y* is protected at compile time by a const modifier. This gives C++ programmers an efficient method of passing large objects while working with their values as opposed to their identities.

Figure 2 **Using a Reference Type**

```
void Foo()
{
  for (int x = 0; x < 10; ++x)
  {
    // Call Bar, passing the identity of x
    Bar(x);
  }
}

void Bar(int& y)
{
  // Display the value of y
  cout << y << " ";

  // Change the value that the identity y refers to
  // Note: This will affect the variable x
  y = y + 1;
}

// Outputs:
// 0 2 4 6 8
```

Figure 3 **The const Modifier**

```
void Foo()
{
  for (int x = 0; x < 10; ++x)
  {
    // Call Bar, passing the identity of x,
    // yet the value of x will be protected by the const
    Bar(x);
  }
}

void Bar(const int& y)
{
  // Use the value of y
  cout << y << " ";

  // Attempt to change the value of what the identity y refers to
  y = y + 1; // This line will not compile because y is const!
}
```

With the const modifier, C++ has immutable data types that resemble those found in most functional programming languages. However, dealing with these immutable data types is difficult. Furthermore, making deep (full) copies of large objects for every small change isn't efficient. Nonetheless, it should be clear that standard C++ has always had a concept of working with values (even if it's not a very pure concept).

Note that the support for value types extends to user-defined types through copy constructors and assignment operators. C++ copy constructors and assignment operators allow user-defined types to make a deep copy of the object. Keep in mind that while C++ copy constructors can be implemented to make a shallow copy, you'll have to make sure the value semantics are preserved.

C++ 11 Support for Functional-Style Programming

C++ 11 brings a number of new tools for functional-style programming. Perhaps most important, C++ now has support for lambdas (also known as closures or anonymous functions). Lambdas allow you to compose your code in ways that wouldn't have been practical before. This functionality was previously available through functors, which are powerful but less practical to use. (Actually, C++ lambdas write anonymous functors behind the scenes.) **Figure 4** shows how lambdas have improved our code with a simple example that uses the C++ standard library (STL).

In this case, the for_each function applies a lambda to each element of a vector. It's important to note that C++ lambdas have been designed to be used inline when possible; thus lambdas can run as fast as handcrafted code.

While C++ is just one of the many imperative languages that now have lambdas, what makes C++ lambdas special is that (similar to functional programming languages) they can preserve the concept of working with values as opposed to identities. While functional programming languages accomplish this by making variables

Figure 4 **Using Lambdas**

```
void Foo()
{
  vector<int> v;
  v.push_back(1);
  v.push_back(2);
  v.push_back(3);

  for_each(begin(v), end(v), [](int i) {
    cout << i << " ";
  });
}

// Outputs:
// 1 2 3
```

Figure 5 **Capturing by Reference**

```
void Foo()
{
  int a[3] = { 11, 12, 13 };

  vector<function<void(void)>> vf;

  // Store lambdas to print each element of an array
  int ctr;
  for (ctr = 0; ctr < 3; ++ctr) {
    vf.push_back([&]() {
      cout << "a[" << ctr << "] = " << a[ctr] << endl;
    });
  }

  // Run each lambda
  for_each(begin(vf), end(vf), [](function<void(void)> f) {
    f();
  });
}

// Outputs:
// a[3] = -858993460
// a[3] = -858993460
// a[3] = -858993460
```

Figure 6 **C++ Syntax for Controlling Lambda Capture**

[]	Don't capture anything (exactly what I wanted in the first lambda example)
[&]	Capture everything by reference (traditional lambda behavior, though not consistent with functional programming's emphasis on values)
[=]	Capture everything by value (while this preserves the concept of values, it limits the usefulness of the lambdas; also, it can be expensive to copy large objects)
[&ctr]	Capture only ctr, and capture ctr by reference
[ctr]	Capture only ctr, and capture ctr by value
[&,ctr]	Capture ctr by value and everything else by reference
[=,&v]	Capture v by reference and everything else by value
[&, ctr1, ctr2]	Capture ctr1 and ctr2 by value and everything else by reference

Figure 7 **Sharing Variables**

```
void Foo()
{
  // Create a shared int
  // (dynamically allocates an integer
  //  and provides automatic reference counting)
  auto sharedInt = make_shared<int>(123);

  // Share the integer with a secondShare
  // (increments the reference count)
  shared_ptr<int> secondShare(sharedInt);

  // Release the pointer to the first integer
  // (decrements the reference count)
  sharedInt = nullptr;

  // Verify the shared int is still alive
  cout << "secondShare = " << *secondShare << endl;

  // Shared int is automatically de-allocated
  // as secondShare falls out of scope and the reference
  // count falls to zero
}

// Outputs:
// secondShare = 123
```

immutable, C++ does it by providing control over the capture. Consider the code in **Figure 5**.

In this code, everything is captured by reference, which is the standard behavior for lambdas in other languages. Yet capturing by reference complicates things unless the variables being captured are immutable. If you're new to working with lambdas, you probably expect the following output from the code:

```
a[0] = 11
a[1] = 12
a[2] = 13
```

However, that's not the output you get—and the program might even crash. This is because the variable ctr is captured by reference, so all of the lambdas use the final value of ctr (that is, 3, the value that made the for loop come to an end) and then access the array beyond its bounds.

It's also worth noting that to keep the ctr variable alive to be used by the lambda outside of the for loop, the ctr variable's declaration has to be lifted out of the for loop. While some languages eliminate the need to lift value type variables to an appropriate scope, that doesn't really solve the problem, which is that the lambda needs to use the value of ctr as opposed to the identity of the variable ctr. (There are workarounds for other languages that involve making an explicit copy to a temporary variable. However, this makes it a bit unclear as to what's going on, and it's error-prone because the original variable is also captured and thus is still available for use.)

As **Figure 6** shows, C++ provides a simple syntax to allow easy control of the lambda's capture, which preserves the concept of working with values.

It's clear from **Figure 6** that the programmer has complete control over how the lambda captures variables and values. However, while this preserves the concept of working with values, it does nothing to make working with complex data structures as values efficient.

Immutable Data Types

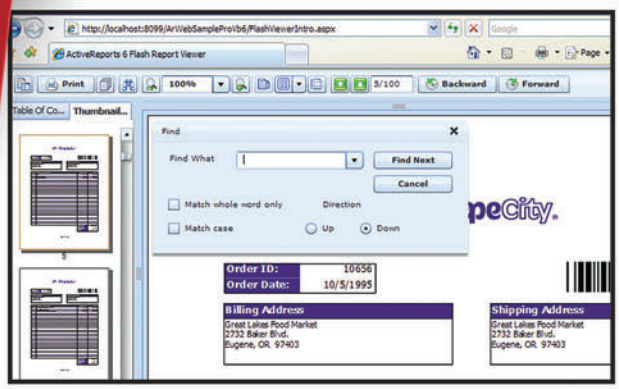
What's missing are the efficient immutable data structures that some functional programming languages have. These languages facilitate immutable data structures that are efficient even when very large because they share common data. Creating data structures in C++ that share data is trivial—you just dynamically allocate data and each data structure has pointers to the data. Unfortunately, it's more difficult to manage the lifetime of shared variables (for this reason, garbage collectors have become popular). Luckily, C++ 11 provides

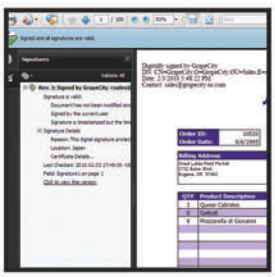
Figure 8 **An Immutable Class for Sharing Data**

```
class Immutable
{
private:
  // Use a normal double, copying is cheap
  double d_;

  // Use a shared string, because strings can be very large
  std::shared_ptr<std::string const> s_;

public:
  // Default constructor
  Immutable()
    : d_(0.0),
      s_(std::make_shared<std::string const>(""))
  {}

  // Constructor taking a string
  Immutable(const double d, const string& s)
    : d_(d),
    s_(std::make_shared<std::string const>(s))
  {}

  // Move constructor
  Immutable(Immutable&& other)
    : s_()
  {
    using std::swap;
    swap(d_, other.d_);
    swap(s_, other.s_);
  }

  // Move assignment operator
  Immutable& operator=(Immutable&& other)
  {
    swap(d_, other.d_);
    swap(s_, other.s_);
    return *this;
  }

  // Use default copy constructor and assignment operator

  // Getter Functions
  double GetD() const
  {
    // Return a copy because double is small (8 bytes)
    return d_;
  }
  const std::string& GetS() const
  {
    // Return a const reference because string can be very large
    return *s_;
  }

  // "Setter" Functions (always return a new object)
  Immutable SetD(double d) const
  {
    Immutable newObject(*this);
    newObject.d_ = d;
    return newObject;
  }
  Immutable SetS(const std::string& s) const
  {
    Immutable newObject(*this);
    newObject.s_ = std::make_shared<std::string const>(s);
    return newObject;
  }
};
```

<activereports>

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





ActiveReports 7
NEW VERSION

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

Follow the ActiveReports 7 road to success

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

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

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

Dani Diaz
Developer Evangelist, Microsoft

EXPECT MORE.
GET MORE.

ComponentOne
a division of GrapeCity

Figure 9 **Using the Smart ImmutableVector Template Class**

```
template <class ImmutableVector>
void DisplayImmutableVector(const char* name, const ImmutableVector& v)
{
  using namespace std;

  cout << name << ".Size() = " << v.Size()
     << ", " << name << "[] = { ";
  for (size_t ctr = 0; ctr < v.Size(); ++ctr) {
    cout << v[ctr] << " ";
  }
  cout << "}" << endl;
}


void ImmutableVectorTest1()
{
  // Create an ImmutableVector with a branching size of four
  ImmutableVector<int, 4> v;

  // Another ImmutableVector (we will take a copy of v at element 6)
  ImmutableVector<int, 4> aCopyOfV;

  // Push 16 values into the vector (this will create a two level tree).
  // Note that the vector is being assigned to itself. The
  // move constructor insures this is not very expensive, but
  // if a copy was made at any point the copy would remain
  // unchanged, but continue to share the applicable data with
  // the current version.
  for (int ctr = 0; ctr < 10; ++ctr) {
    v = AppendValue(v, ctr);
    if (ctr == 6) aCopyOfV = v;
  }

  // Display the contents of the vectors
  DisplayImmutableVector("v", v);
  DisplayImmutableVector("aCopyOfV", aCopyOfV);
}

// Outputs:
// v.Size() = 10, v[] = { 0 1 2 3 4 5 6 7 8 9 }
// aCopyOfV.Size() = 7, aCopyOfV[] = { 0 1 2 3 4 5 6 }
```

an elegant solution for working with shared variables through the std::shared_ptr template class, as shown in **Figure 7**.

The code in **Figure 7** illustrates a simple use of std::shared_ptr and its helper function std::make_shared. Using std::shared_ptr makes it easy to share data among data structures without fear of leaking memory (as long as circular references are avoided). Note that std::shared_ptr provides the basic thread-safety guarantees, and runs fast because it uses a lock-free design. However, keep in mind that the basic thread-safety guarantee that std::shared_ptr provides doesn't automatically extend to the object to which it's pointing. Still, std::shared_ptr guarantees it will not reduce the thread-safety guarantee of the object it points to. Immutable objects inherently provide a strong thread-safety guarantee because once they're created they never change. (Actually, they never change in an observable manner, which includes, among other things, an appropriate thread-safety guarantee.) Therefore, when you use a std::shared_ptr with an immutable object, the combination maintains the immutable object's strong thread-safety guarantee.

I can now easily create a simple immutable class that potentially shares data, as shown in **Figure 8**.

The code in **Figure 8** is a bit long, but most of it is boilerplate code for the constructors and assignment operators. The last two functions are the key to making the object immutable. Note that the SetS and SetD methods return a new object, which leaves the original object unchanged. (While including the SetS and SetD methods as members is convenient, it's a bit of a lie, because they don't actually change the original object. For a cleaner solution, see the ImmutableVector in **Figures 9** and **10**.) **Figure 11** shows the Immutable class in action.

Note that object *b* shares the same string as object *a* (both strings are at the same address). Adding additional fields with associated getters and setters is trivial. Though this code is good, it's a little more difficult to scale to containers when you're being efficient. For example, a naïve ImmutableVector might maintain a list of shared pointers representing each element of the array. When the naïve ImmutableVector is changed, the entire array of shared pointers would need to be duplicated, incurring additional cost as each shared_ptr element of the array would need its reference count to be incremented.

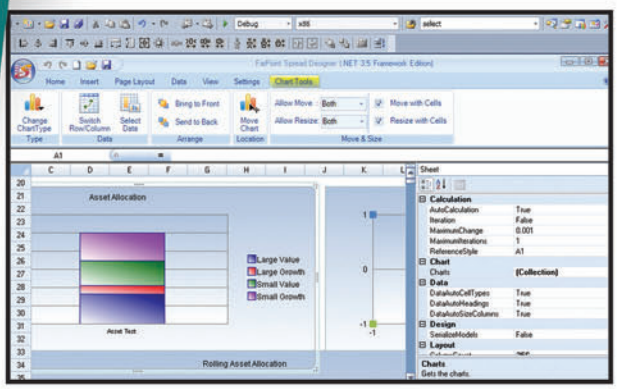
There is a technique, though, that allows the data structure to share most of its data and minimize the duplication. This technique uses a tree of some form to require duplication of only the nodes that are directly affected by a change. **Figure 12** shows a comparison of a naïve ImmutableVector and a smart ImmutableVector.

This tree technique scales nicely: as the number of elements grows, the percent of the tree that needs to be duplicated is minimized. Moreover, by adjusting the branching factor (so each node has more than two children), you can achieve a balance in memory overhead and node reuse.

Figure 10 **Methods for Operating on the ImmutableVector**

```
void ImmutableVectorTest2()
{
  ImmutableVector<int, 4> v;
  v = AppendValue(v, 1);
  v = AppendValue(v, 2);
  v = AppendValue(v, 3);
  int oldValue = v.Back();
  auto v1 = TruncateValue(v);
  auto v2 = SubstituteValueAtIndex(v, 0, 3);
  auto v3 = GenerateFrom(v, [](ImmutableVector<int, 4>::MutableVector& v) {
    v[0] = 4;
    v[1] = 5;
    v[2] = 6;
    v.PushBack(7);
    v.PushBack(8);
  });
  auto v4 = GenerateFrom(v3, [](ImmutableVector<int, 4>::MutableVector& v4) {
    using namespace std;
    cout << "Change v4 by calling PopBack:" << endl;
    cout << "x = v4.PopBack()" << endl;
    int x = v4.PopBack();
    cout << "x == " << x << endl;
    cout << endl;
  });

  // Display the contents of the vectors
  DisplayImmutableVector("v", v);
  DisplayImmutableVector("v1", v1);
  DisplayImmutableVector("v2", v2);
  DisplayImmutableVector("v3", v3);
  DisplayImmutableVector("v4", v4);
}

// Outputs:
//    Change v4 by calling PopBack:
//    x = v4.PopBack()
//    x == 8
//
//    Resulting ImmutableVectors:
//    v.Size() = 3, v[] = { 1 2 3 }
//    v1.Size() = 2, v1[] = { 1 2 }
//    v2.Size() = 3, v2[] = { 3 2 1 }
//    v3.Size() = 5, v3[] = { 4 5 6 7 8 }
//    v4.Size() = 4, v4[] = { 4 5 6 7 }
```

<spread>

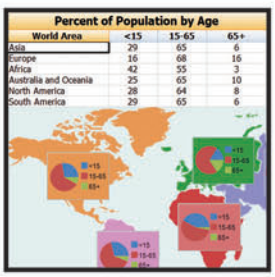


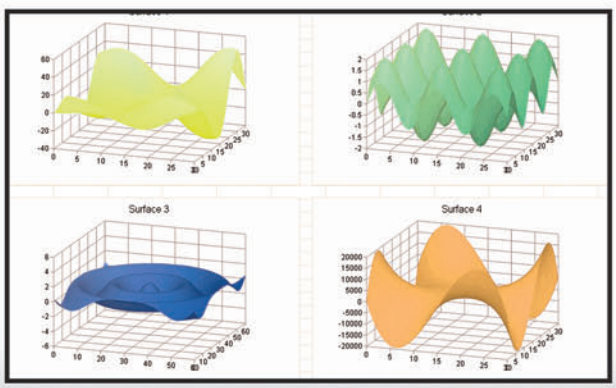
Spread's powerhouse components make it as easy as 1-2-3:

Import Microsoft Excel documents, preserving complete formatting

Interact with the data in Spread from within your application

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





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

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

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

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

Jeffrey Baron
VP App Software Development

EXPECT MORE.
GET MORE.

ComponentOne
a division of GrapeCity

Figure 11 **The Immutable Class in Action**

```
using namespace std;

void Foo()
{
  // Create an immutable object
  double d1 = 1.1;
  string s1 = "Hello World";
  Immutable a(d1, s1);

  // Create a copy of the immutable object, share the data
  Immutable b(a);

  // Create a new immutable object
  // by changing an existing immutable object
  // (Note the new object is returned)
  string s2 = "Hello Other";
  Immutable c = a.SetS(s2);

  // Display the contents of each object
  cout << "a.GetD() = " << a.GetD() << ", "
    << "a.GetS() = " << a.GetS()
    << " [address = " << &(a.GetS()) << "]" << endl;
  cout << "b.GetD() = " << b.GetD() << ", "
    << "b.GetS() = " << b.GetS()
    << " [address = " << &(b.GetS()) << "]" << endl;
  cout << "c.GetD() = " << c.GetD() << ", "
    << "c.GetS() = " << c.GetS()
    << " [address = " << &(c.GetS()) << "]" << endl;
}

// Outputs:
// a.GetD() = 1.1, a.GetS() = Hello World [address = 008354BC]
// b.GetD() = 1.1, b.GetS() = Hello World [address = 008354BC]
// c.GetD() = 1.1, c.GetS() = Hello Other [address = 008355B4]
```

I developed a smart ImmutableVector template class that can be downloaded from archive.msdn.microsoft.com/mag201208CPP. **Figure 9** shows how you can use my ImmutableVector class. (As previously noted, to make the immutable nature of the ImmutableVector clearer to the users, ImmutableVector uses static member functions for all actions that generate new versions.)

For read-only actions, the vector can be used much like a regular vector. (Note that for this example I haven't implemented iterators, but doing so should be fairly trivial.) For write actions, the AppendValue and TruncateValue static methods return a new ImmutableVector, thus preserving the original object. Unfortunately, this isn't reasonable for the array subscript operator, so I made the array subscript operator read-only (that is, it returns a const reference) and provided a SubstituteValueAtIndex static method. It would be nice, however, to be able to make a large number of modifications using the array subscript operator in a single block of code. To facilitate this, ImmutableVector provides a GenerateFrom static method, which takes a lambda (or any other functor). The lambda in turn takes a reference to MutableVector as a parameter, which allows the lambda to work on a temporary MutableVector that can be changed freely like a normal std::vector. The example in **Figure 10** shows the various methods for operating on the ImmutableVector.

The beauty of the GenerateFrom static method is that the code within it can be written in an imperative way, while resulting in an immutable object that can be safely shared. Note that the GenerateFrom static method prevents unauthorized access to the underlying ImmutableVector by disabling the MutableVector it passed into the lambda as soon as the lambda exited. Please note as well that while ImmutableVector provides a strong thread-safety guarantee, its helper class MutableVector does not (and is intended to be only used locally within the lambda, not passed around to other threads). My implementation also optimizes for multiple changes within the Change method such that there's minimal restructuring occurring on the temporary tree, which gives a nice performance boost.

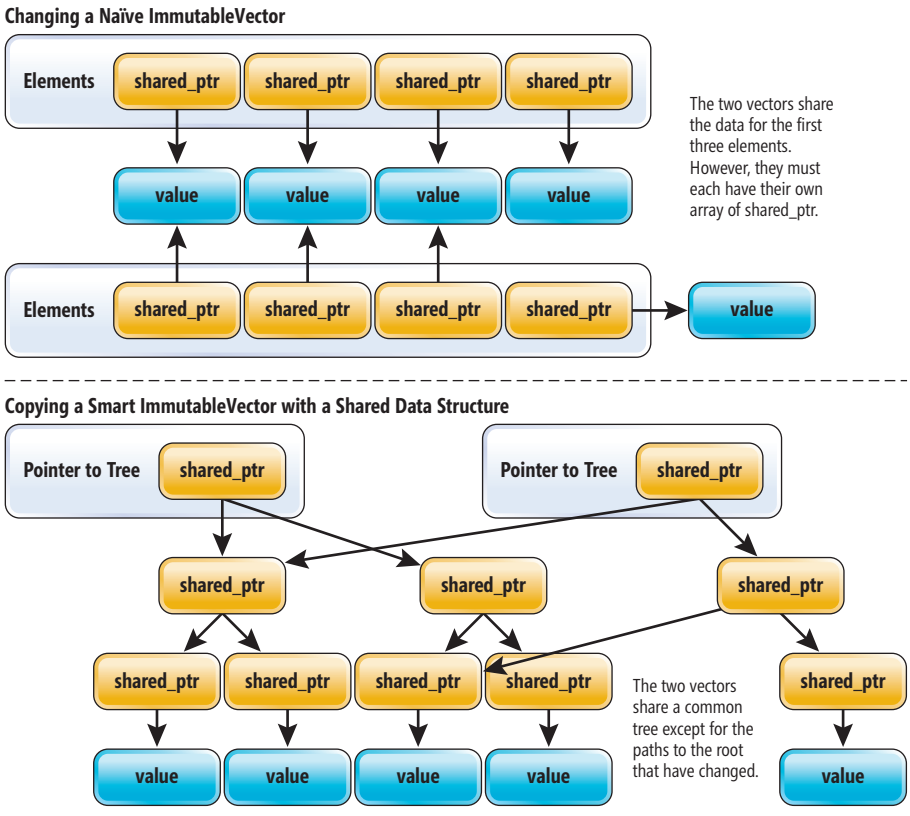


Figure 12 **Comparing Naïve and Smart ImmutableVectors**

Wrapping Up

This article gives you just a taste of how you can use functional-style programming in your C++ code. Moreover, C++11 features such as lambdas bring a touch of functional-style programming regardless of the paradigm used. ■

DAVID CRAVEY *is a Visual C++ MVP who enjoys programming in C++ maybe a bit too much. You'll find him presenting at local C++ user groups and universities. During the day he enjoys working at NCR, through TEKsystems in Fort Worth, Texas.*

THANKS to the following technical experts for reviewing this article: *Giovanni Dicanio, Stephan T. Lavavej, Angel Hernández Matos, Alf P. Steinbach and David Wilkinson*

TECHMENTOR CONFERENCES

AUGUST 20-24
MICROSOFT HQ
REDMOND, WA

Powered by Microsoft | TechNet

Only one event in 2012!

IN-DEPTH TRAINING FOR IT PROS

This year, TechMentor is a **ONE-TIME-ONLY** event at a new special location – Microsoft Headquarters!

Receive cutting-edge and practical education for the IT professional! Learn from IT experts and industry insiders on topics such as:

* Windows 8 Server
* MCITP Training
* Security
* Windows PowerShell
* Virtualization
* Cloud Computing

. . . and much more!

* Visit the Microsoft Campus in Redmond, Washington
* Receive Unbiased, Immediately Usable and In-Depth Tech Training
* Network with Peers, IT Experts and Microsoft Insiders

techmentorevents.com

Space is Limited — Register Today!

Use promo code AUG1AD

Visit techmentorevents.com or scan the QR code to register and for more event details.

Produced By: 1105 MEDIA Inc

Supported By: Redmond MAGAZINE | Microsoft | TechNet | VIRTUALIZATION REVIEW

Windows Azure Comes to the Rescue

Mark Kromer

Our marketing department in the Microsoft Mid-Atlantic District hosted an event last spring for about 90 Fortune 500 executives, Microsoft partners and Microsoft employees. They needed a registration system to highlight the latest and greatest Microsoft technologies and immerse attendees in the newest Microsoft experience. I had built successful solutions in my previous roles as a solution architect and consultant, so they turned to me in a pinch. This is because there wasn't yet any clear direction on what technologies to showcase, or how. Oh, and—by the way—the conference was just a month away!

This article discusses:

- Solution architecture
- Windows Azure Web apps
- Windows Phone app
- Windows Azure SQL Database and Windows Azure SQL Reporting Services

Technologies discussed:

Windows Azure, Windows Azure SQL Database, Windows Azure SQL Reporting, Windows Azure WCF Services, Windows Phone, Silverlight, Windows Azure BLOB Storage

Code download available at:

archive.msdn.microsoft.com/mag201208Azure

My focus at Microsoft is to work with large businesses on solutions that are built on SQL Server and Windows Azure SQL Database, known as the Microsoft data platform. So I immediately gravitated toward the Platform as a Service (PaaS) capabilities of Windows Azure as the only way I could possibly meet marketing's requirements. By leveraging the ability of Windows Azure to quickly deliver a solution built on the Microsoft application platform (the Microsoft .NET Framework, SQL Server and Windows Server), I could make this happen. With the Windows Azure PaaS model, Microsoft handles the infrastructure, power, servers, maintenance, patching, upgrades and so on from worldwide Microsoft datacenters, making this a zero-footprint, zero-infrastructure option, which was perfect for my requirements.

Part of the agreement with marketing and with my manager was that I was only going to devote my spare time and weekends to this project. So I turned to already well-known technologies proven for quick time to production, which included Silverlight, Windows Communication Foundation (WCF) and SQL Server. In the end, I was able to deliver a solid solution that resulted in positive feedback and good results, using common Microsoft technologies and with no budget—and without a large time investment. I hope you'll gain some guidance and insights from the techniques I leveraged with Windows Azure, Windows Azure SQL Database and Windows Phone development, and minimal effort.

For this project I primarily needed to focus on two specific requirements. One was to make it easy and compelling for attendees to

interact with new devices and equipment from the conference partners such as tablets, slates and phones. The second was to highlight the new Metro design approach Microsoft is using across all devices. I didn't have time to load up the community preview of Windows 8 and Visual Studio 11 to make a native Metro application. But I had been developing on Windows Phone 7.5 for several months, so I decided to go hybrid with a Silverlight approach because it would work for both Windows 8 desktop mode and Windows Phone native applications.

Figure 1 **The PivotViewer Control Utilized as a Tile Interface for Registered Attendees**

To meet the requirements, I decided to create an attendee check-in application for registered guests to interact with when they arrived at the registration desk. That covered the requirement of a functional and interactive application. To create a Metro UI experience, I utilized the Silverlight 5.0 PivotViewer control. In keeping with the interactive themes and presentation of different form-factor experiences, I created a Windows Phone app based on the Panorama control. With the app, the Microsoft employees at the event could allow guests to utilize their devices to check in or view event photos. Guests who were also Windows Phone users could download the app from the Marketplace. An easy and obvious technology choice was Windows Azure, and I used Windows Azure SQL Database for the database, Windows Azure SQL Reporting for tracking reports, Windows Azure WCF Services for data exchange and Windows Azure Binary Large Object (BLOB) Storage for storing on-site event photos.

Solution Overview

I'm going to split up the solution architecture into three primary areas for this article: the Windows Azure Web apps; Windows Azure SQL Database and reporting; and the Windows Phone app. The complete solution also included a few other ancillary, nice-to-have apps and features that I won't detail in the limited space I have here. But if you're looking at building a custom solution for a conference or event on this scale (fewer than 300 attendees), these other areas are important for you to consider:

1. **Photo uploading and sharing:** I utilized the image uploader tool from the Windows Azure Toolkit, which takes images from local storage and uploads them to Windows Azure BLOB Storage. It works great out of the box. You just need to plug in your Windows Azure Storage key and modify the Default.aspx file to your own design. Images were uploaded so users could share photos from the event through a photo gallery on large screens at the event or on demo tables. I recommend downloading a tool to explore the files in your Windows Azure Storage so you can then manage those files easily after uploading. One such tool, the Azure Storage explorer, is available on CodePlex at bit.ly/H3r0C.
2. **Social networking:** This is a requirement for an app of this nature. For public conferences and events, make sure you include a way to share those photos as well as attendee comments, Tweets, updates, conversations and so on. In the Windows Phone app I simply linked to a Facebook group that I created, and in the Web app (which I'll describe later) I used the Facebook C# SDK for ASP.NET, which you can get at bit.ly/J5D2zI.
3. **Silverlight photo gallery viewer:** I won't fully describe this now, but I chose to use this as a quick fix for the requirement of displaying images on an overhead screen in the main conference room. As the images were dropped

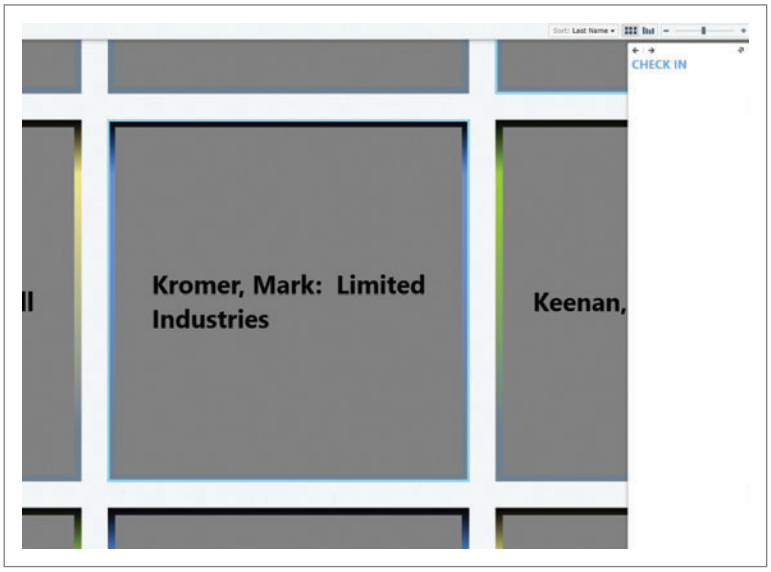


Figure 2 **The PivotViewer Zooms in on a Selected Name So the User Can Check In**

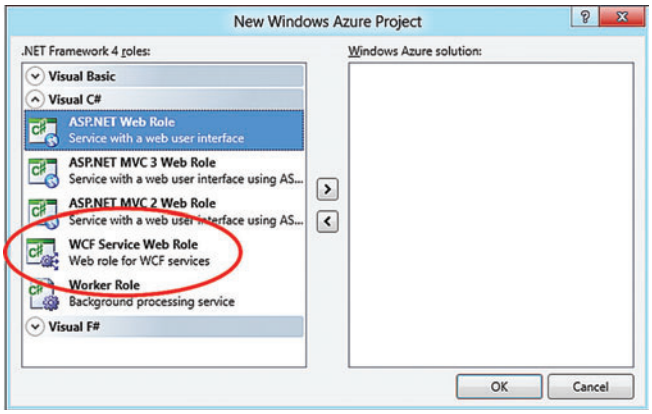


Figure 3 **The WCF Service Web Role in Windows Azure**

into Windows Azure Storage, this page could refresh manually and display an endless loop, or display on laptops with which users could interact. The control and XML configurations came from CodePlex (slideshow.codeplex.com). There are many other photo viewers that you can use. The important criterion, if you go this route with Windows Azure Storage, is that the component must be able to render images from a Web URL. In previous use of Windows Azure Storage, I used plain "http://" links directly to JPG and PNG graphics files.

I had a limited runway in which to work for my time to production, and Windows Azure provided a way to ramp up quickly by using all these different puzzle pieces. This is because the PaaS offering is made up of .NET, Windows Server, Windows Azure Storage and Windows Azure SQL Database. That meant my experience in .NET, SQL Server and Windows Server worked essentially

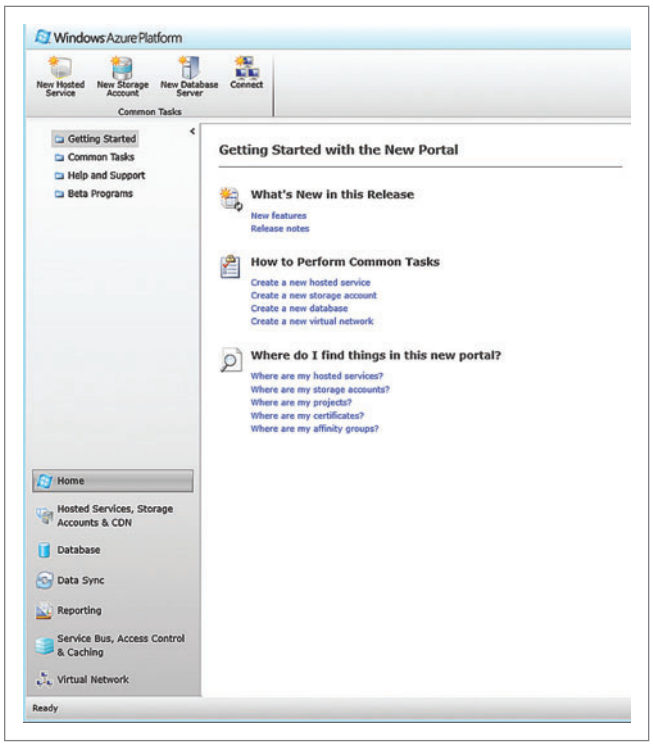


Figure 4 **The Windows Azure Management Console**

the same for me once I learned the nuances of adding a Windows Azure project to my Visual Studio 2010 solutions. I'll touch on some of these later in the solution description. Finally, because this was a one-time event, I didn't need to purchase infrastructure. Instead, I deployed the components to my Windows Azure account and switched things off when I was done. I have a SQL Saturday

Figure 5 **Using WCF Service to Interact with Windows Azure SQL Database**

```
namespace WCFServiceConference
{
  public class Service1 : IService1
  {
    // MyData is used to retrieve the attendee list
    public List<String> MyData(int value)
    {
      List<String> results = new List<String>();

      using (retreatEntities context = new retreatEntities())
      {
        IQueryable<Register> sortedContacts = context.registers
          .OrderBy(c => c.lastname)
          .ThenBy(c => c.org);

        // Register is the EF class

        foreach (Register sortedContact in sortedContacts)
        {
          results.Add(sortedContact.lastname + ", " + sortedContact.email +
            "    (" + sortedContact.firstname + ") : " + sortedContact.org);
        }
      }

      return results;
    }


    // Use this method to check in attendees and check out
    public void ToggleRegister(String lname, String fname)
    {
      using (retreatEntities context = new retreatEntities())
      {
        IQueryable<register> sortedContacts = context.registers;

        try
        {
          Register qry = (from register in context.Registers
                          where Register.lastname == @lname &&
                          Register.firstname == @fname
                          select Register).First();

          // checkinis a binary SQL Server field, so I'm using byte array to set it
          byte[] s1 = qry.checkin;

          // We're going to toggle the checkin value where 1 == checked-in
          if (s1[0] == 0)
          {
            s1[0] = 1;

            // Set the check-in date/time field in SQL
            qry.checkdate = DateTime.Now.ToString();
          }
          else
          {
            s1[0] = 0;
            qry.checkdate = null;
          }

          qry.checkin = s1;

          // Now let's save our changes
          context.SaveChanges();
        }
        catch (Exception) { }
      }
    }
  }
}
```

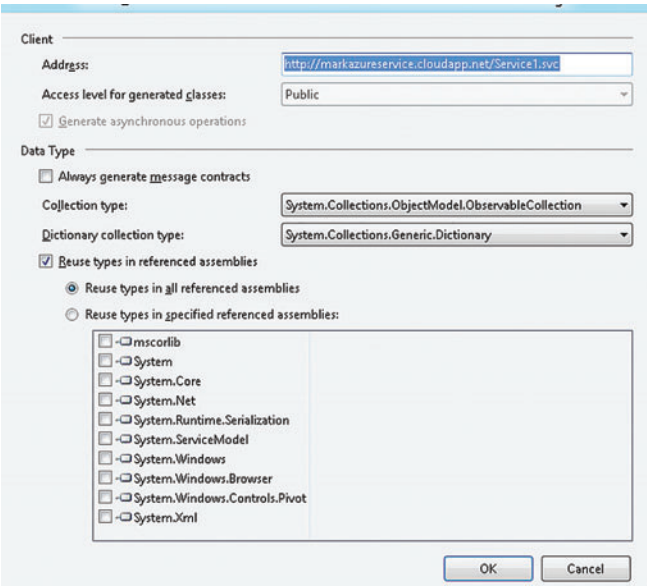


Figure 6 **Referencing the Windows Azure-Hosted WCF Service in the Web App**

event coming up this summer and will likely spin these components back up when the time comes around to reuse this conference event system. Because Windows Azure bills monthly like a power utility would, I'm only billed based on my utilization, and I can scale to my user requirements without needing to over-provision infrastructure for worst-case scenarios.

Windows Azure Web Apps

I'll start with the primary entry screen for the registration app, which is a C# .NET Web app hosting a Silverlight 5.0 XAML control, hosted in Windows Azure. Something that you'll find different when developing for Windows Azure instead of traditional .NET projects in Visual Studio is the framework for the hosting services in Windows Azure that will host WCF services and Web apps. There

Figure 7 **Receiving Updates Through Data Binding in the Silverlight Component via a WCF Service**

```
public static Service1Client _sc;

// This init method is in my MainPage.xaml file
// and is called right after InitializeComponent()
public static void init()
{
  _sc = new Service1Client();
  _sc.MyDataCompleted +=
    new EventHandler<MyDataCompletedEventArgs>(_sc_MyDataCompleted);
  _sc.MyDataAsync(1);
}

public static void _sc_MyDataCompleted(object sender, MyDataCompletedEventArgs e)
{
  // How many rows were returned?
  snamescnt = e.Result.Count;

  snames = new List<string>();

  for (int i = 0; i < snamescnt-1; i++)
  {
    snames.Add(e.Result[i].ToString());
  }
}
```

are custom classes in my C# class in my top-level namespace to handle converting data coming from Windows Azure SQL Database to Silverlight properties. In the Web tier, you'll see the classic XML files (for example, crossdomain.xml) to allow cross-domain calls from the Web tier (that is, Silverlight) to a back-end data service. This Web app now becomes the primary role in my Windows Azure service because I selected it as the Web role in my project.

For this app, I utilized one of my favorite Silverlight controls from Windows Live labs (which is now a part of Silverlight 5.0) called the PivotViewer control.

The PivotViewer control is generally applicable for application scenarios where you need to enable easy navigation and an interactive UI for large amounts of data that can be easily categorized, sorted and navigated by the end user (see **Figure 1**).

> To bind data from the Windows Azure SQL Database to the Silverlight front end, I created a WCF service hosted in Windows Azure.

The PivotViewer also enables customization of user actions so that when attendees find their names in the tiles, they can then check in by clicking on their named tiles (see **Figure 2**). The color-based tile UI for PivotViewer was chosen by the organizers of the Microsoft conference because it represents a similar theme to the Windows 8 Metro interface and Windows Phone tile interface. This user experience worked well for the registration desk for the conference, and the tiles made for perfect interactive touch mechanisms for users to self-check-in on large touchscreen monitors. Additionally, the control has the built-in capability to change the sort order based on your data's properties, so the names could be easily arranged by last name.

To modify the appearance of the tiles in PivotViewer, you need to add attributes to the PivotViewerItemTemplate. Notice how I added a color converter and set the attendee name with the "ShortName" attribute:

```
<conv:TextToSolidColorConverter x:Key="colorConverter"/>
<pivot:PivotViewerItemTemplate x:Key="DemoTemplate">
  <Border Width="300" Height="300" Background="{Binding Color,
    Converter={StaticResource colorConverter}}">
  <TextBlock Text="{Binding ShortName}" FontWeight="ExtraBold"
    FontSize="42"HorizontalAlignment="Center" VerticalAlignment="Center"
    TextWrapping="Wrap" />
 </Border>
</pivot:PivotViewerItemTemplate>
```

The color is set from nothing more than a loop through the list of attendees and then modification of the item number property. The XAML for the color is set in the converter class:

```
var xaml = "<LinearGradientBrush xmlns='http://schemas.microsoft.com/client/2007'
  StartPoint=\"0.5,0\" EndPoint=\"0.5,1\">" +
  "<GradientStop Color=\"Black\" Offset=\"0.0\" />" +
  "<GradientStop Color=\""+value.ToString()+"\" Offset=\"0.15\" />" +
  "<GradientStop Color=\"SlateGray\" Offset=\"0.85\" />" +
  "</LinearGradientBrush>";
```

To bind data from the Windows Azure SQL Database to the Silverlight front end, I created a WCF service hosted in Windows

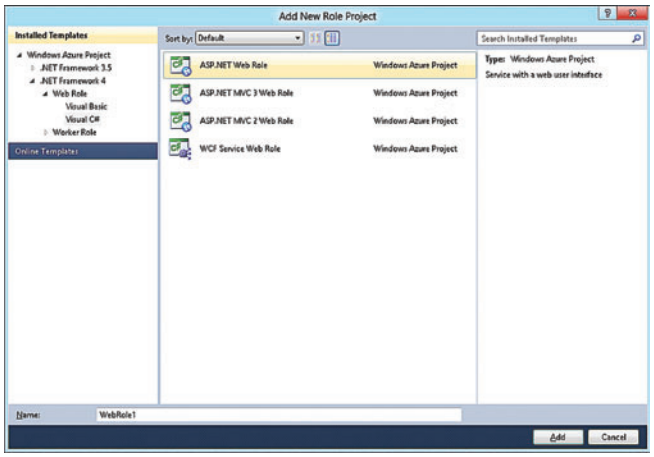


Figure 8 **Adding an ASP.NET Web Role for Simple Web Pages**

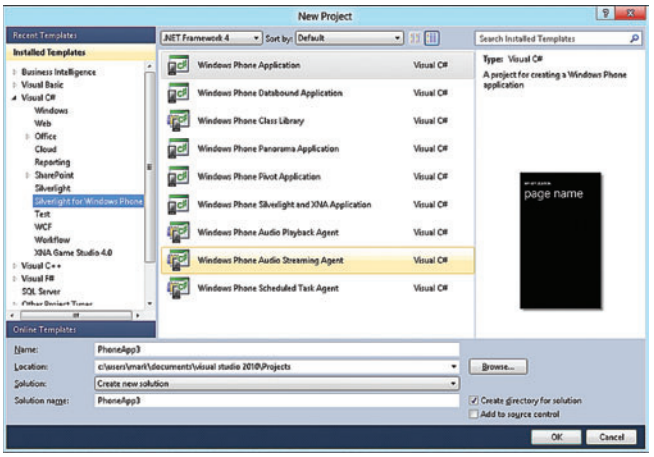


Figure 9 **The Windows Phone Silverlight App Project Template**

Azure. When you develop Web applications on Windows Azure, you have the option of storing your WCF Web services on your Windows Azure account. I created my WCF service for the specific purpose of acting as the middle tier to my solution, exposing database data from Windows Azure SQL Database. **Figure 3** shows what you select when creating a new WCF service for Windows Azure in Visual Studio 2010.

When deploying a solution on Windows Azure, you'll need to access different areas of the Windows Azure management console (see **Figure 4**). In the case of the PivotViewer Web App, the WCF service and the check-in Web app, all components are deployed and stored under the "Hosted Services" section. Later in this article, I'll use the Database, Reporting and Storage sections as well. Deploying a Web application—in Windows Azure lingo—is known as creating a hosted service with an ASP.NET Web Role, whereas the WCF data service that I created for accessing the Windows Azure SQL Database is also a hosted service, but with a WCF Service Web Role instead. In each case, you'll receive a cloudapp.net domain name with your custom service name prepended to the URL. To use your own domain name, you'll need to map your domain name to the Windows Azure-friendly DNS name using a technique explained in the MSDN Library article, "How to Configure a Custom Domain for a Windows Azure Hosted Service" (bit.ly/MfSBaD).

The WCF service is a straightforward Web service to query data from Windows Azure SQL Database to report the attendee information. It also includes a method to update the status of the attendee. By selecting his name on the PivotViewer tiles, the user toggles his status of checked in or checked out.

To access a database in Windows Azure SQL Database, you use a connection string that's really no different than accessing a classic SQL Server database. Being able to move code, snippets and previous code to Windows Azure is incredibly easy because the connection types and mechanisms are the same. In the code shown in **Figure 5**—from my WCF service where I access the database in Windows Azure SQL Database—I'm utilizing my Entity Framework Register class, which connects to Windows Azure SQL Database in the same way as you would connect to any other SQL Server database.

When you publish your services to Windows Azure, you can then reference them in your Visual Studio projects just as you normally would through service references (see **Figure 6**), making Windows Azure a great place to house your public Web services.

Now that I have the cloud database—and the cloud services to interact with the data—completed and published, I can interact with the services from my Silverlight PivotViewer app by using an asynchronous service. Because the interaction with the data service is asynchronous, you'll have to set up a corresponding local event method to capture the data when it arrives and populate the client component.

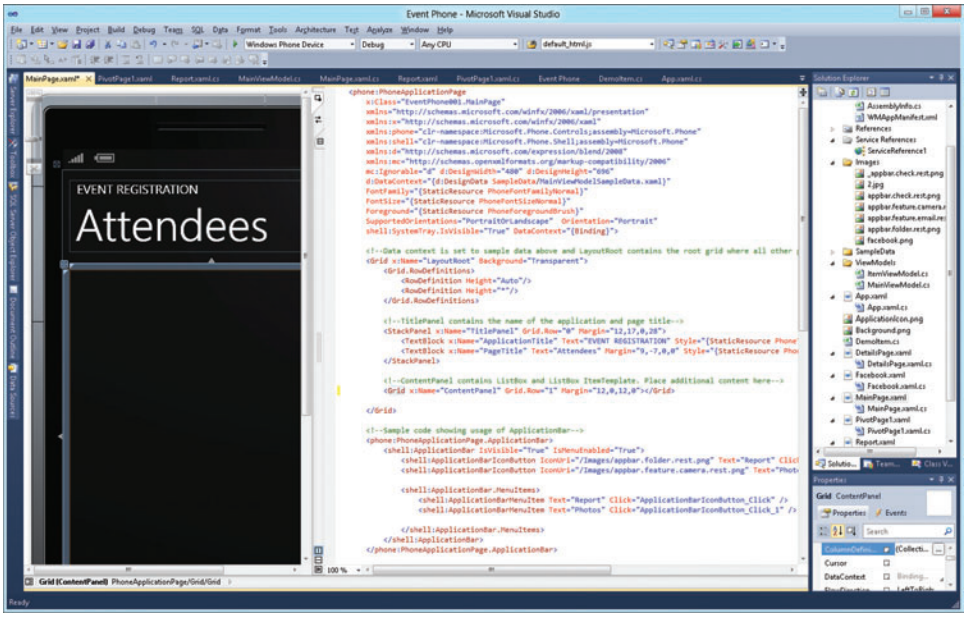


Figure 10 **Bind Data to Silverlight in Windows Phone Apps in XAML the Same Way as in a Silverlight App**

DevExpress

Dazzling Design.
Developer Delivered.

Beautiful design and rich user experiences are moving to the center of the development conversation. Your existing applications now must run on an array of mobile devices, like iPads, slates and mobile phones. Building applications designed for the device they'll run on requires more focus than ever on great design principles . The design time visualization, dynamic themes and Touch-enabled tools in **DXperience 12.1** by DevExpress help you realize your vision.

Your users are ready.
Ensure you're ready, too.

Download your free 30-day trial at DevExpress.com

DXv2

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

u e win asp wpf sl cr

This is just a List of strings in my case. When the *snames* property of my class is updated, the Silverlight component will receive the updates through data binding, as shown in **Figure 7**.

The solution also includes a small ASP.NET Web app that displays confirmation, issues the check-in request and links to Facebook so attendees can link together via a Facebook group. The calls to the WCF service to update the database are much more straightforward because ASP.NET includes server-side capabilities that Silverlight does not. The following code shows how easy it is to utilize the service:

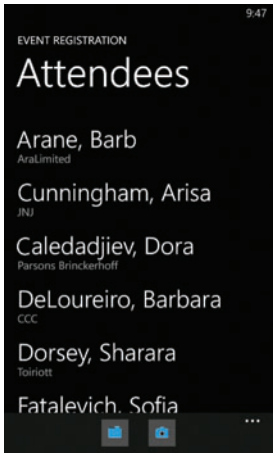


Figure 11 **Windows Phone App with Attendee List and Menu Button Navigation**

```
String lname = Request["lname"];
String fname = Request["fname"];
// No need to use async event calls from ASP.NET
// Just reference the Web service and call the method directly
Service1Client sc = new Service1Client();
sc.ToggleRegister(lname, fname);
Label1.Text = "CHECKING IN " + fname.ToUpper() + " " + lname.ToUpper();
```

This ASP.NET page was then deployed directly to my Windows Azure account as a Web Role (see **Figure 8**).

The Windows Phone App

Windows Phone apps are built using Silverlight with the Silverlight for Windows Phone project template in Visual Studio 2010 (see **Figure 9**). Because the app is Silverlight, I was able to quickly generate a second form-factor application on Windows Phone to complement the Web app described previously. This is because the same techniques I used to bind the Silverlight client controls in XAML are also used to connect to the WCF service—hosted in Windows Azure—that I used for the Web app, making the service highly reusable.

I made the Windows Phone app fully functional for the conference, meaning that you could view all registered attendees, run reports on attendees and allow Facebook and e-mail interactions. So, to allow check in—as opposed to the ASP.NET approach described previously—I needed asynchronous event callback methods for both retrieving the data from Windows Azure and updating the rows:

```
public MainViewModel() {
  this.Items = new ObservableCollection<ItemViewModel>();
  _sc = new Service1Client();

  // Same as earlier in the Silverlight PivotViewer app
  _sc.MyDataCompleted +=
    new EventHandler<MyDataCompletedEventArgs>(_sc_MyDataCompleted);

  // Used to update the row
  _sc.CheckInCompleted +=
    new EventHandler<CheckInCompletedEventArgs>(_sc_CheckInCompleted);

  // Get the data
  _sc.MyDataAsync(1);
}
```

That technique is the same one used in Silverlight Windows apps, as I demonstrated earlier. However, in this case, I utilized the ViewModel class.

I utilized the ListBox control in a Windows Phone Portrait Page (see **Figure 10**) as my MainPage, which loads when the app is loaded on the Windows Phone (see **Figure 11**).

I retrieved the data from Windows Azure SQL Database and populated the ListBox from the ViewModel by iterating through each row returned from my Windows Azure WCF service with the results of the query:

```
public void _sc_MyDataCompleted(object sender, MyDataCompletedEventArgs e)
{
  for (int i = 0; i < e.Result.Count - 1; i++)
          {
  // Parse the incoming text for attendee name, e-mail address and company name
  string strMain = e.Result[i].ToString();
  string strName = strMain.Substring(0, strMain.IndexOf('(') - 1);
  string strEmail = strMain.Substring(
    strName.Length + 2, strMain.IndexOf(')')-strName.Length-2);
  string strCompany = strMain.Substring(strMain.IndexOf(':') + 2);
  this.Items.Add(new ItemViewModel()
    {LineOne=strName, LineTwo=strCompany, LineThree=strEmail });
}
```

One of the popular features of the Windows Phone app that I published for conference events is the photo viewer. At the beginning of this article, I mentioned that I provided an image-upload tool that took images and persisted them in Windows Azure BLOB Storage. To display those on Windows Phone, I simply added another page (Windows Phone navigation uses a Web-based page paradigm) that users could navigate to from a menu button on the bottom of the main page (see **Figure 11**). This is simple. Just add something like the following to the button click event on a menu item:

```
NavigationService.Navigate(new Uri("/PivotPage1.xaml", UriKind.Relative));
```

For the photo gallery page, I used the Pivot (yes, there's that pivot word again) page type in Windows Phone, which allows users to smoothly scroll through a list of items. The Pivot control takes a list of PivotItems in XAML that you use to create a smooth user experience while scrolling horizontally. In my case, I just had to add a series of Web Browser controls as Photo 1, Photo 2, Photo 3 and so on, as shown in **Figure 12**.

Because of the time and resource constraints on the project, I kept it simple, with the Windows Phone browser control pointing to the URI for Windows Azure BLOB Storage for each photo. This

Figure 12 **Adding a Series of Web Browser Controls**

```
<controls:Pivot Title="Microsoft Executive Technology Retreat">
  <controls:PivotItem Header="Photo 1">
    <Grid>
      <phone:WebBrowser HorizontalAlignment="Left" Margin="6,3,0,0"
        Name="MyWebBrowserControl" VerticalAlignment="Top" Height="605"
        Width="452" />
    </Grid>
  </controls:PivotItem>
  <controls:PivotItem Header="Photo2">
    <Grid>
      <phone:WebBrowser HorizontalAlignment="Left" Margin="6,3,0,0"
        Name="MyWebBrowserControl1" VerticalAlignment="Top" Height="605"
        Width="452" />
    </Grid>
  </controls:PivotItem>
...
  <controls:PivotItem Header="Photo10">
    <Grid>
      <phone:WebBrowser HorizontalAlignment="Left" Margin="6,3,0,0"
        Name="MyWebBrowserControl9" VerticalAlignment="Top" Height="605"
        Width="452" />
    </Grid>
  </controls:PivotItem>
</controls:Pivot>
```

allowed the user interaction to include resizing, which doesn't come naturally in the Windows Phone Image control.

The coding for the e-mail piece to send an attendee an e-mail note was also simple. It used the built-in EmailComposeTask API in Windows Phone:

```
private void ApplicationBarEmailButton_Click(object sender, EventArgs e)
{
  EmailComposeTask emailComposeTask = new EmailComposeTask();
  emailComposeTask.To = ContentText.Text;
  emailComposeTask.Body = "Hi There!";
  emailComposeTask.Subject = "Microsoft Executive Technology Retreat";
  emailComposeTask.Show();
}
```

This API uses the existing contact list on the user's phone (see **Figure 13**).

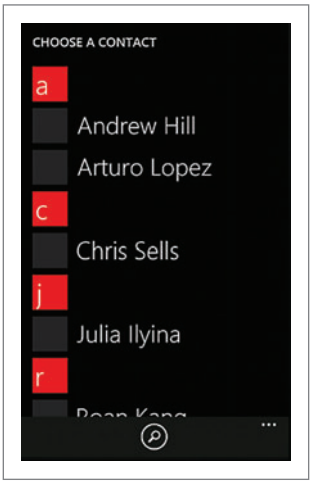


Figure 13 **The Windows Phone E-Mail Contact List**

Windows Azure SQL Database and Windows Azure SQL Reporting Services

The database I created for this quick-and-dirty solution was simple. Because I used Windows Azure SQL Database, I was able to build the schema directly in classic SQL Server on-premises tools including SQL Server Management Studio, SQL Server Data Tools and Business Intelligence Development Studio.

There's one aspect to this solution that isn't yet complete: the online registration sign-up portion, which was handled by an existing Microsoft-hosted solution. Interestingly, for a technology company, there wasn't something readily available to provide the on-site interactive experience for the attendees, which is what I'll show you how to do here. That being said, the way that the database in Windows Azure SQL Database was populated for this project was through SQL Server Integration Services (SSIS), as shown in **Figure 14**. This is a simple, straightforward SSIS package that takes an Excel dump of the registered users and maps them to fields in my database schema in Windows Azure SQL Database. The Derived Column transform is being used here to modify data in some fields that might not perfectly match my schema. In this kind of solution, this is an area that will probably need to be a bit more complex than what I'm showing you here. But what I want you to take away from this is that in a cloud-based solution such as this Windows Azure event system, common tools such as SSIS work perfectly well, so you won't need to learn new tools.

What you should be concluding by now is that the current developer and data-management tools for Windows Azure make a solution such as this a hybrid solution. Another area that also demonstrates the Microsoft move into cloud computing is Windows Azure SQL Reporting, which I utilized to provide up-to-the-minute reports on who had checked into the conference with timestamps and contact information (see **Figure 15**). Anyone who has built any type of SQL Server Reporting Services (SSRS) reports in the past will be familiar with this format. Essentially, Windows Azure SQL Reporting is SSRS in the cloud, running on the Windows Azure platform. I really like the fact that Windows Azure SQL Reporting is purely Web-based, without the need for Silverlight. This meant I was also able to render this exact same report in Windows Phone without needing to recreate a report for the mobile app. I just referenced it from a Web Browser control:

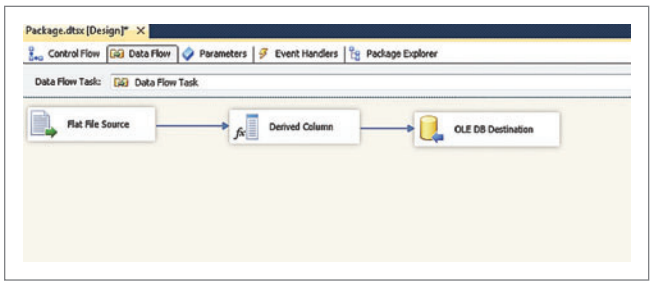


Figure 14 **Using SQL Server Integration Services**

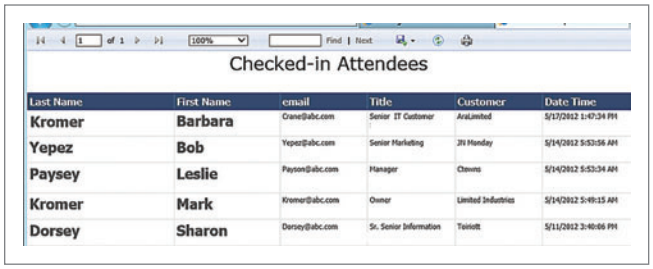


Figure 15 **Windows Azure SQL Reporting Was Used to Provide Reports on Attendee Check-Ins**

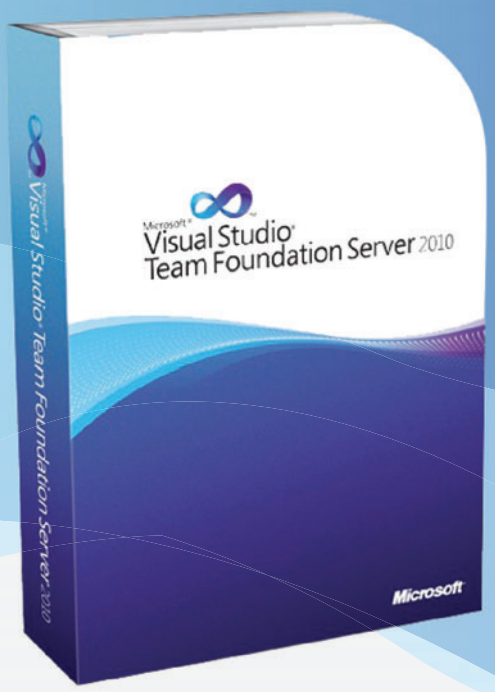
```
webBrowser1.Navigate(new
  Uri("https://fxnxxajmxx.reporting.windows.net/
ReportServer?%2fAttendees&rs:
  Command=Render&rc:Toolbar=false", UriKind.Absolute));
```

Notice that the Web service for Windows Azure SQL Reporting is identical to an on-premises SSRS service, meaning I can use URL parameters to modify the report appearance. **Figure 15** shows the Web page view that the volunteers at the check-in table viewed on their laptops and Windows Phones. As attendees arrived and checked in, this report let them find the correct stickers and badges for each attendee from behind the check-in desk.

So, that's pretty much it. There were quite a few moving parts because I needed to quickly glue together a solution using as much existing technology as possible that allowed for quick configuration and very little infrastructure. This made for a perfect use case to utilize cloud computing. Using Windows Azure for storage, hosting and Web pages allowed me to stick with ASP.NET, a WCF service and a cloud database with Windows Azure SQL Reporting (which I was already familiar with from an on-premises experience). And sticking with Silverlight as a UI technology for the Web pages allowed me to reuse the same techniques for multiple form factors quickly and easily. ■

Mark Kromer *is a Microsoft SQL Server technology specialist based in Philadelphia. He was previously a Microsoft BI Solutions product manager for Microsoft and Oracle Corp., as well as a software engineer and database consultant for AT&T and Agilent Technologies Inc.*

Thanks to the following technical expert for reviewing this article: *David Ateek*



VISUAL STUDIO TEAM FOUNDATION SERVER 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 2010/2008 Integration

TFS HOSTING ADDONS & MIGRATION SERVICES

- TFS Build Server
- UrbanTurtle Scrum Tools
- VSS to TFS Migration
- TFS 2010 to Hosted TFS Migration

LIVE! 360

IT EVENTS WITH PERSPECTIVE

December 10-14, 2012 | Royal Pacific Resort, Orlando, FL

CODE IN THE SUNSHINE!

Visual Studio Live! is back in Orlando, bringing attendees everything they love about this event: hard-hitting, practical .NET Developer training from the industry's best speakers and Microsoft insiders. But this year, there's an extra bonus – THREE extra co-located events that you can attend for free: SharePoint Live!, SQL Server Live! and Cloud & Virtualization Live! **live360events.com**

Who Should Attend?

- Developers
- IT Pros
- Database Administrators
- System Architects
- Integrators
- Engineers
- Systems Administrators

GOLD SPONSOR axosoft SHIP SOFTWARE ONTIME

SUPPORTED BY Microsoft

SAVE UP TO $400 –
Register Before October 10th

Use Promo Code DEVAUG

WHAT IS LIVE! 360?

LIVE! 360 is a new event brought to you by the publishers of *Visual Studio* Magazine, *MSDN Magazine*, *Redmond* Magazine, *Redmond Channel Partner* Magazine, and *Virtualization Review* Magazine; and the producers of Visual Studio Live! and TechMentor conferences. This event will bring together Visual Studio Live!, an established and respected independent software conference, with three new industry events: Cloud & Virtualization Live!, SharePoint Live!, and SQL Server Live!.

Live360events.com

Scan the QR code or visit live360events.com for more information.

LIVE! 360

IT EVENTS WITH PERSPECTIVE

Buy 1 Event, Get 3 Free!

Customize an agenda to suit YOUR needs – attend just one Live! event or all four for the same low price!

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**

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**

SQL Server LIVE!

TRAINING FOR DBAs AND IT PROS

Bringing SQL Server to Your World
IT Professionals, DBAs and Developers – across a breadth of experience and organizations – come together for comprehensive education and knowledge share for SQL Server database management, performance tuning and troubleshooting. **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**

Microsoft Visual Studio | msdn | Visual Studio MAGAZINE | Redmond MAGAZINE | VIRTUALIZATION REVIEW

PRODUCED BY 1105 MEDIA

MEDIA SPONSOR THE CODE PROJECT WWW.CODEPROJECT.COM

Build User-Friendly XML Interfaces with Windows PowerShell

Joe Leibowitz

The Windows PowerShell scripting language does everything you want a command-line tool to do—and so much more—that it could eventually replace technologies such as VBScript. For a good general description of what Windows PowerShell is about and the basics of using it, see bit.ly/LE4SU6 and bit.ly/eBucBI.

Windows PowerShell is thoroughly integrated with the Microsoft .NET Framework and thus is deeply connected to XML, the current international standard for data exchange using structured text files. For general information about XML, see bit.ly/JHfzw.

This article explores the capacity of Windows PowerShell to present and manipulate XML data with the goal of creating a relatively simple UI for reading and editing XML files. The idea is to make this easier and more convenient using algorithms that "understand" the sibling and parent-child relations of any given file, even without foreknowledge of the schema. I'll also examine the use of Windows Forms in Windows PowerShell, XPath querying and other related technologies. The proposed app can digest an XML file and emit XPath queries on its own.

This article discusses:

- Creating an XML parsing engine
- Constructing the Windows Forms
- Using XPath queries to search or edit an XML file

Technologies discussed:

Windows PowerShell, Windows Forms, XML, XPath

Code download available at:

archive.msdn.microsoft.com/mag201208PowerShell

Let's look at how you can analyze any XML file in Windows PowerShell and present it in a format that people without advanced technical skills can use. **Figure 1** shows a preview of the type of GUI you can create.

The key to making this happen is to enable the Windows PowerShell application to parse and understand any XML file without human guidance or foreknowledge of its schema. After researching existing technologies for automated analysis of XML files, I decided to develop a parsing engine for this specific purpose, because what I was able to find didn't fully address the need to understand XML documents without human viewing. Currently, applications universally seem to assume that a developer or user is well-acquainted with the elements, attributes and overall schema of any given XML document. But some—possibly many—situations in the real world fall outside this paradigm. For example, in a scenario with many data consumers who aren't XML experts but who need access to a variety of XML data sources, the familiarity assumption of the existing paradigm fails. Similarly, even with a trained expert or two on staff, if an organization confronts hundreds or thousands of differently structured XML files, human handling could easily become overwhelmed.

Therefore, what's needed is a parsing engine that will read any XML file and emit XPaths that ordinary users, with only a minimum of training, can use to search and edit any given XML file.

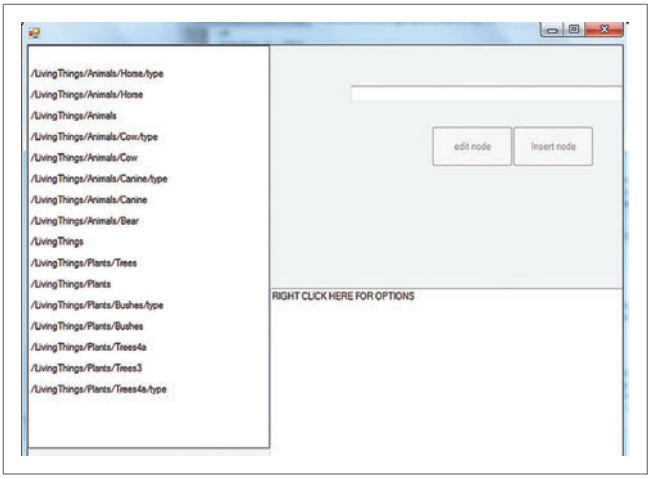


Figure 1 **Preliminary View of the GUI**

The XML Parsing Engine

To be compliant XML, a document's closing and opening brackets must match. For example, if an element <ABC> exists, there must also exist at some later point in the same file an element </ABC>. Between these opening and closing angle brackets, almost anything can theoretically occur. Using this fundamental principle of XML, I'll show you how to automatically construct a comprehensive series of XPath queries such that even relatively inexperienced XML data consumers can quickly put them to use to find and manipulate data in XML files.

> To be compliant XML, a document's closing and opening brackets must match.

First, establish a set of arrays to hold all opening and closing brackets in the XML file:

```
[int[]]$leading_brackets = @()
[int[]]$closing_brackets = @()
[string[]]$leading_value = @()
[string[]]$closing_value = @()
```

To build a strongly typed array of unknown size in Windows PowerShell, three elements are necessary: the [type[]] leading part; a $name part; and the symbol for an array of unknown size, @(). Variables in Windows PowerShell take $ as their leading character. These particular arrays cover the indexed locations of opening and closing angle brackets in the XML document as well as the string values of the element names associated with these brackets. For example, in the XML line <PS1>text value</PS1>, the integer index of the leading brackets would be 0 and the index of the closing brackets would be 15. The leading and closing values in this case would be PS1.

To get our target XML into memory, we use the following code:

```
$xdoc = New-Object System.Xml.XmlDocument
      $xdoc.Load("C:\temp\XMLSample.xml")
```

Figure 2 is a partial view of the actual XML file being used.

After the load operation, this XML data is in memory. In order to manipulate and analyze the XML, I use the document object model that's now instantiated in the $xdoc variable (but I'll also need to use the XPathNavigator technology for a few special purposes, as noted later in this article):

```
# Create an XPath navigator (comments in PowerShell code take the "#"
leading character)
$nav = $xdoc.CreateNavigator()
```

One of the most interesting features of Windows PowerShell is the built-in function, or cmdlet, called Get-Member, which lets you examine the methods and properties of any object in Windows PowerShell right in the IDE as you develop. **Figure 3** includes a call to this cmdlet on the $nav object just created, and **Figure 4** shows the results displayed in the Windows PowerShell Integrated Scripting Environment (ISE) when the Get-Help call is made.

While Get-Member will often put you on the right track during Windows PowerShell development, you'll also find the associated Get-Help cmdlet handy during this process.

If you type get-help xml at the command line, as shown in **Figure 4**, you'll get the output shown here:

```
getName              Category  Synopsis
-------              --------  --------
Export-Clixml        Cmdlet    Creates an XML-based representation of an object or...
Import-Clixml        Cmdlet    Imports a CLIXML file and creates corresponding obj...
ConvertTo-XML        Cmdlet    Creates an XML-based representation of an object.
Select-XML           Cmdlet    Finds text in an XML string or document.
about_format.ps1xml  HelpFile The Format.ps1xml files in Windows PowerShell defin...
about_types.ps1xml   HelpFile Explains how the Types.ps1xml files let you extend ...
```

If you type get-help about_types.ps1xml, you'll see the results shown in **Figure 5**.

The Windows PowerShell integrated system for researching syntax is comprehensive and relatively easy to use. This is a topic worthy of its own article.

Figure 2 **Partial View of the Sample XML File**

```
<?xml version="1.0" encoding="utf-8"?>
<Sciences>
  <Chemistry>
    <Organic ID="C1" origination="Ancient Greece" age="2800 yrs">
      <branch ID="C1a">
        <size>300</size>
        <price>1000</price>
        <degree>easy&gt;</degree>
        <origin>Athens</origin>
        // Text for organic chem here
      </branch>
      <branch name="early" ID="C1b" source="Egypt" number="14">
        <size>100</size>
        <price>3000</price>
        <degree>hard&gt;</degree>
        <origin>Alexandria</origin>
        // Text for original Egyptian science
      </branch>
    </Organic>
  </Chemistry>
<Physics></Physics>
<Biology ID="B" origination="17th century" >
.
.
.

      <Trees4a name="trees4a" age="40000000">
        <type ID="Tda1">oakda</type>
        <type ID="Tda2">elmda</type>
        <type ID="Tda3">oakd3a</type>
      </Trees4a>
    </Plants>
  </Biology>
</Sciences>
```

Figure 3 **Results of Get-Member Call**

```
                    Get-Member -InputObject $nav
                  TypeName: System.Xml.DocumentXPathNavigator

Name                MemberType Definition
----                ---------- ----------
AppendChild         Method     System.Xml.XmlWriter AppendChild(), System.V...
AppendChildElement  Method     System.Void AppendChildElement(string prefix...
CheckValidity       Method     bool CheckValidity(System.Xml.Schema.XmlSche...
Clone               Method     System.Xml.XPath.XPathNavigator Clone()
ComparePosition     Method     System.Xml.XmlNodeOrder ComparePosition(Syst...
Compile             Method     System.Xml.XPath.XPathExpression Compile(str...
CreateAttribute     Method     System.Void CreateAttribute(string prefix, s...
CreateAttributes    Method     System.Xml.XmlWriter CreateAttributes()
CreateNavigator     Method     System.Xml.XPath.XPathNavigator CreateNaviga...
DeleteRange         Method     System.Void DeleteRange(System.Xml.XPath.XPa...
DeleteSelf          Method     System.Void DeleteSelf()
Equals              Method     bool Equals(System.Object obj)
Evaluate            Method     System.Object Evaluate(string xpath), System...
GetAttribute        Method     string GetAttribute(string localName, string...
GetHashCode         Method     int GetHashCode()
TypeName: System.Xml.DocumentXPathNavigator

.
.
.
.
.
Value               Property   System.String Value {get;}
ValueAsBoolean      Property   System.Boolean ValueAsBoolean {get;}
ValueAsDateTime     Property   System.DateTime ValueAsDateTime {get;}
ValueAsDouble       Property   System.Double ValueAsDouble {get;}
ValueAsInt          Property   System.Int32 ValueAsInt {get;}
ValueAsLong         Property   System.Int64 ValueAsLong {get;}
ValueType           Property   System.Type ValueType {get;}
XmlLang             Property   System.String XmlLang {get;}
XmlType             Property   System.Xml.Schema.XmlSchemaType XmlType {get;}
```

To get the XML into analysis-ready condition, use the Select method of XpathNavigator:

```
$nav.Select("/") | % { $ouxml = $_.OuterXml }
```

In the first part of this statement, I invoke .Select on the simple XPath query "/", giving the entire XML contents. In the second part, after the Windows PowerShell symbol | for its object pipeline, I do a foreach, represented by the alias %; I could've used foreach rather than the alias. Inside the loop, I build the working XML string data variable $ouxml from the .OuterXML property of the objects being processed in the loop. Referring back to **Figure 3**, .OuterXML is one of the properties of the XPathNavigator object. This property provides a complete set of all the angle brackets in the XML file, which is required for the parsing engine to work properly.

Note that for objects going through a pipeline, $_ is the symbol for the particular instance, with dot notation used to obtain each instance's properties and methods. Every object in the pipeline is addressed or referenced using the $_ symbol. To get an attribute of the $_ object, use, for example, $_.Name (if Name is a member property of the particular object). Everything passing through a Windows PowerShell pipeline is an object with properties and methods.

A last preparation stage before parsing is to "regularize" the XML text by treating any special cases that look like <ShortNode/>. The parsing engine would rather see the same information in a different format: <ShortNode></ShortNode>. The following code starts this transformation using a RegEx and looking for matches:

```
$ms = $ouxml | select-string -pattern "<([a-zA-Z0-9]*)\b[^>]*/>"   -allmatches
foreach($m in $ms.Matches){ 'regularize' to the longer format }
```

You can now look at the main analytical code for this application: the parsing engine that will populate the four arrays listed earlier. **Figure 6** shows code that tests the file for opening brackets.

The code in **Figure 6** handles the special case of the root element of the XML document. Another fundamental rule of XML is that every schema should contain a single overall root set of angle brackets; inside of these enclosing symbols, the XML data can be structured in any manner consistent with the matching rule mentioned earlier, that is, for every "<ABC>" there's an "</ABC."

Notice that the += syntax is used to add an item or element to an array. Later, after being populated with elements, such an array can be accessed via indexing, as in $leading_brackets[3].

In the IndexOf arguments, note that the starting position for the search, represented by the second parameter in the method, shows a reference to $Script:ctr. In Windows PowerShell, variables have distinct scopes that follow from where they're created. Because the variable $ctr here is created outside the scope of any function, it's considered script-level, and a script-level variable can't be changed from inside a function without referring to $Script. Inside a loop, rather than inside a function, the $Script reference may not be required, but it's a good habit to keep scope in mind at all times.

When coding, a good clue to a scope violation is a variable that should be changing in value but isn't; usually, this is because it's out of scope and needs to be prefixed accordingly.

Once the root element is handled, all other elements are handled within one else block:

```
else
{
# Check for more "<"
$check = $ouxml.IndexOf("<",$leading_brackets[$Script:ctr-1]+1)
if($check -eq - 1)
{
break
}
```

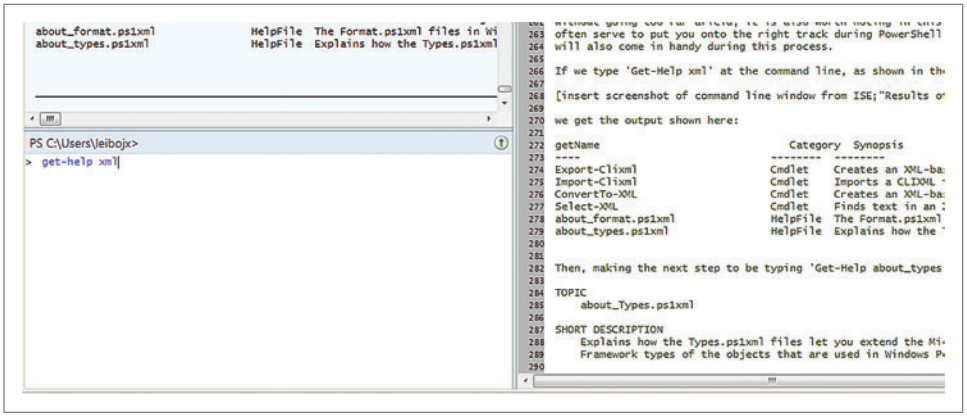


Figure 4 **Results of Get-Help in Windows PowerShell**

Figure 5 **Getting Help with Types.ps1xml Files**

```
TOPIC
  about_Types.ps1xml

SHORT DESCRIPTION
  Explains how the Types.ps1xml files let you extend the Microsoft .NET
  Framework types of the objects that are used in Windows PowerShell.

LONG DESCRIPTION
  The Types.ps1xml file in the Windows PowerShell installation directory
  ($pshome) is an XML-based text file that lets you add properties and
  methods to the objects that are used in Windows PowerShell. Windows
  PowerShell has a built-in Types.ps1xml file that adds several elements to
  the .NET Framework types, but you can create additional Types.ps1xml
  files to further extend the types.

SEE ALSO
  about_Signing
  Copy-Item
  Get-Member
  Update-TypeData
```

The first thing to do is to check whether the end of the file has been reached; the criterion for that event is the absence of further < symbols. The preceding code does this. If there are no more < symbols, a break is called.

The next segment of code distinguishes between < cases and </ cases:

```
#eliminate "</" cases of "<"
if($ouxml.IndexOf("</",$leading_brackets[$Script:ctr-1]+1) -ne `
  $ouxml.IndexOf("<",$leading_brackets[$Script:ctr-1]+1))
```

Notice the Windows PowerShell syntax for "not equal" in comparisons: -ne. Related operators include -eq,-lt and -gt.

Because you're trying to accumulate all the opening angle brackets, you want to know only about these at this stage of parsing engine operations. Notice the Windows PowerShell syntax for "not equal" in comparisons: -ne. Related operators include -eq,-lt and -gt. Also, as in Visual Basic (but unlike C#), you need a line-wrapping character, the back-tick symbol (`), to continue a line of code.

If the test succeeds, populate the $leading_brackets array with a new element:

```
$leading_brackets += $ouxml.IndexOf("<",$leading_brackets[$Script:ctr-1]+1)
```

With the newest iteration of leading angle brackets established, the next task is to isolate the name of the associated element. For this task, note that after the initial opening < and element name, <ElementName, there's either a space followed by one or more attributes, or the brackets close, as in the following two cases:

```
<ElementName attribute1="X" attribute2 = "Y">, or
<ElementName>
```

Separate these two cases with the following code, which looks to see which comes first, a space or the > symbol:

```
$indx_space = $ouxml.IndexOf(" ",$leading_brackets[$Script:ctr])
  $indx_close = $ouxml.IndexOf(">",$leading_brackets[$Script:ctr])
  if($indx_space -lt $indx_close)
  {
  $indx_to_use = $indx_space
  }
  else
  {
  $indx_to_use = $indx_close
  }
```

Once you establish the proper ending point, employ $indx_to_use to help isolate the string associated with the leading angle bracket that's now in focus:

```
$leading_value += $ouxml.
Substring($leading_brackets[$Script:ctr],($indx_to_use -
  $leading_brackets[$Script:ctr]))
```

In effect, the leading value is the string starting with < and ending with either a space or a >.

The stage is set to pick up the correlative closing angle brackets by finding the string </ElementName:

```
$closing_brackets += $ouxml.IndexOf("</" + $leading_value[$Script:ctr].
Substring(1),`
  $leading_brackets[$Script:ctr]+1)
$Script:ctr+=1
```

Finally, in case the distinction between < and </ is not met, increment the array element and continue:

```
else
{
$leading_brackets[$Script:ctr-1] +=1
}
```

At the end of this process, the three arrays look like the following partial presentation of their data:

$leading_brackets:
0 18 62 109 179 207 241 360 375 447 475 509 625 639 681 713 741 775 808 844 900 915 948 976 1012 1044 1077 1142 1154 1203 1292 1329 1344 1426 1475 1490 1616 1687 1701 1743 1810 1842 1890 1904 1941 1979 2031 2046 2085 2138 2153 2186 2235 2250 2315 2362 2378 2442 2476 2524 2539 2607 2643 2718

$leading_value:
<Sciences <Chemistry <Organic <branch <size <price <degree <origin <branch <size <price <degree <origin <Physics <Biology

$closing_brackets:
2718 1687 625 360 179 207 241 273 612 447 475 509 541 1142 900 713 741 775 808 844 882 1129 948 976 1012 1044 1077 1

Figure 6 **Testing a File for Opening Brackets**

```
# if you run out of "<" you're done, so use the "$found_bracket"
Boolean variable to test for presence of "<"

$found_bracket = $true

while($found_bracket -eq $true)
{
  # Special case of first, or root element, of the XML document;
  # here the script-level variable $ctr equals zero.
    if($Script:ctr -eq 0)
    {
    #to handle the top-level root
    $leading_brackets += $ouxml.IndexOf("<",$leading_brackets[$Script:ctr])
    $leading_value += $ouxml.Substring(0,$ind)
    $closing_brackets += $ouxml.IndexOf("</" + $leading_value[0].Substring(1))

    $Script:ctr+=1
    }
}
```

Figure 7 **Checking for Duplicate Xpaths**

```
$is_dupe = $false
  foreach($xp in $xpaths)
  {
  $depth = $xpath.Length
  $xp = $xp.Replace('/////','')
  $xpath2 = $xpath
  $xpath2 = $xpath2.Replace(" ","")
  $xpath2 = $xpath2.Replace("<","")
  if($xp -eq $xpath2)
  {
  $is_dupe = $true
  #write-host 'DUPE!!!'
  break
}
```

Figure 8 **Adding Buttons and Textboxes**

```
$pointA = New-Object System.Drawing.Point
$listbox = New-Object Windows.Forms.Listbox
$form.Controls.Add($listbox)
$listbox.add_SelectedIndexChanged({PopulateTextBox})
$form.Controls.Add($button_get_data)
$form.Controls.Add($text_box)
$pointA.X = 800
$pointA.Y = 100
$button_get_data.Location = $pointA
$button_get_data.Width = 100
$button_get_data.Height = 50
$pointA.X = 400
$pointA.Y = 50
$text_box.Location = $pointA
$text_box.Width = 800
```

Establishing Nodal Relationships

Now it's time for the second phase in parsing engine operations. In this more complex phase, the sequences of $leading_brackets and $closing_brackets establish the parent-child and sibling relations among all the nodes of the XML being parsed. First, a number of variables are established:

```
# These variables will be used to build an automatic list of XPath queries
$xpaths = @()
$xpaths_sorted = @()
$xpath = @()
[string]$xpath2 = $null
```

Next, a first pairing of adjacent leading and closing brackets is fixed:

```
$first_open = $leading_brackets[0]
$first_closed = $closing_brackets[0]
$second_open = $leading_brackets[1]
$second_closed = $closing_brackets[1]
```

And some loop counters are created:

```
$loop_ctr = 1
$loop_ctr3 = 0
```

The engine will parse iteratively no more times than the value of the $ctr variable incremented during the first phase when building the $leading_brackets and other arrays (the following if statement is the litmus test in terms of establishing the nodal structure of the XML):

```
if($second_closed -lt $first_closed)
```

If the $second_closed value is less than (-lt) the $first_closed value, a child relationship is established:

```
<ElementOneName>text for this element
  <ChildElementName>this one closes up before its
parent does</ChildElementName>
</ElementOneName>
```

With a child node detected, the variables are reset to the next two adjacent pairs of opening-closing angle brackets, the counters are incremented and the vital $xpath array is populated with a new element:

```
$first_open = $leading_brackets[$loop_ctr]
$first_closed = $closing_brackets[$loop_ctr]
$second_open = $leading_brackets[$loop_ctr + 1]
$second_closed = $closing_brackets[$loop_ctr + 1]
$loop_ctr2 +=1
#if($loop_ctr2 -gt $depth){$loop_ctr2 -= 1}
$depth_trial+=1
$xpath += '/' + $leading_value[$loop_ctr-1]
$loop_ctr+=1
```

You've now reached the critical processing stage for the parsing engine: What to do when the parent-child relation no longer holds.

> With all of the automatic XPath queries in hand, you're ready to build a Windows Forms app.

A preliminary matter is to eliminate duplicates that will arise in the course of parsing engine operations. To do this, the variable holding the entire array of XPath queries (which is the key value constructed by the parsing engine) is reviewed element by element to ensure that it doesn't already contain the new proposed candidate for inclusion in $xpaths, which at this point is the current value of $xpath, established in the eighth line of the code in **Figure 7**.

If the current $xpath value is not a duplicate, it's appended to the $xpaths array and $xpath is recreated as an empty array for its next use:

```
if($is_dupe -eq $false){$xpaths += ($xpath2 + '/////');}
$xpath = @()
$xpath2 = $null
```

The essential device used by the parsing engine to continue through the XML iteratively is to rebuild the arrays at each iteration. To accomplish this, the first step is to create new interim array objects as transitional devices:

```
$replacement_array_values = @()
$replacement_array_opens = @()
$replacement_array_closes = @()
$finished = $false
$item_ct = 0
```

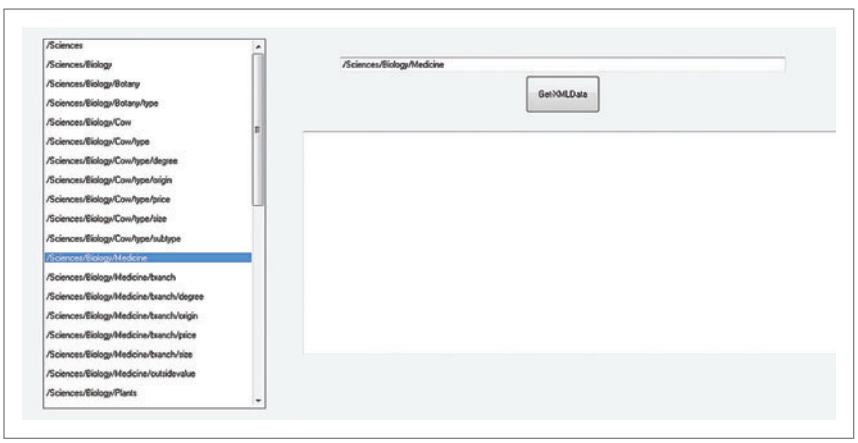


Figure 9 **Selecting an XPath Query**

The engine loops through the $leading_value array and filters out just the current one:

```
foreach($item in $leading_value)
{
if($item -eq $leading_value[$loop_ctr - 1] -and $finished -eq $false)
{
$finished = $true
$item_ct+=1
continue  #because this one should be filtered out
}
```

Unfiltered values are populated into the interim array. All three arrays are populated via association because the array of element name values corresponds in its indexing with the opening and closing angle bracket arrays:

```
$replacement_array_values += $item
$replacement_array_opens += $leading_brackets[$item_ct]
$replacement_array_closes += $closing_brackets[$item_ct]
$item_ct +=1
```

When the three interim arrays are complete, the three permanent arrays are assigned their new values:

```
$leading_value = $replacement_array_values
  $opening_brackets = $replacement_array_opens
  $closing_brackets = $replacement_array_closes
  $loop_ctr+=1
```

> It's worth noting that add_Click is Windows PowerShell syntax for attaching an event to a control.

The next iteration of the first phase of the parsing engine is readied by initializing the first adjacent pairs of angle brackets:

```
$first_open = $leading_brackets[0]
$first_closed = $closing_brackets[0]
$second_open = $leading_brackets[1]
$second_closed = $closing_brackets[1]
$loop_ctr = 1
$loop_ctr2 = 1
continue  # Sends the engine back to the top of the loop
```

Finally, to complete the set of XPath queries, you generate short paths that the previously described process might not have included. For instance, in the current example, without this extra last step, the XPath \Sciences\Chemistry would not be included. The underlying logic is to test that every shorter version of every XPath query also exists, without duplicates. The function that performs this step is AddMissingShortPaths, which you can see in the code download for this article (archive.msdn.microsoft.com/mag201208PowerShell).

With all of the automatic XPath queries in hand, you're ready to build a Windows Forms app for users. In the meantime, the XPath queries just produced are put into the file C:\PowerShell\XPATHS.txt via the Windows PowerShell >> output syntax.

Constructing the Windows Forms Application

Because Windows PowerShell hosts .NET libraries and classes, you can write the following code and thereby make available to your application Windows Forms and the Drawing classes of .NET:

```
[void] [Reflection.Assembly]::LoadWithPartialName("System.Windows.Forms")
[void] [System.Reflection.Assembly]::LoadWithPartialName("System.Drawing")
```

With these basic building blocks in place, you can build a form and its controls, as follows:

```
$form= New-Object Windows.Forms.Form
$form.Height = 1000
$form.Width = 1500
$drawinfo = 'System.Drawing'
$button_get_data = New-Object Windows.Forms.button
$button_get_data.Enabled = $false
$text_box = New-Object Windows.Forms.Textbox
$button_get_data.Text = "get data"
$button_get_data.add_Click({ShowDataFromXMLXPathFilter})
```

It's worth noting that add_Click is Windows PowerShell syntax for attaching an event to a control—in this case, attaching a function call to the button's click event. The code in **Figure 8** adds buttons and textboxes.

In order to populate $listbox with your collection of XPath queries, do the following:

```
foreach($item in $xpaths)
{
$listbox.Items.Add($item.Substring(0,$item.Length - 5))
# Each row in the listbox should be separated by a blank row
$listbox.Items.Add('     ')
}
```

The UI

Figure 9 displays the UI with the XPath queries generated by the tool shown on the left, one of which was selected by the user.

In the final step, the user presses the GetXMLData button and produces the results shown in **Figure 10**.

There you have it—a simple UI for reading and editing XML files, created entirely with Windows PowerShell. In upcoming *MSDN Magazine* online articles, I'll continue on this subject by showing you how to handle XML files that use namespaces, as well as demonstrate the use of the techniques shown here to allow editing of XML files via the interface. ■

Figure 10 **The Results Window**

JOE LEIBOWITZ *is a consultant who specializes in infrastructure projects. He can be reached at joe.leibowitz@bridgewaresolutions.com.*

THANKS to the following technical expert for reviewing this article: *Thomas Petchel*

A History (API) Lesson

Clark Sell

History, in the context of a Web browser, has generally meant the back button, and it has never been easy to manage. It became even more of a challenge when AJAX became significant in Web applications and we started to see the typical Web site grow into more of a rich Web application. And as JavaScript became enabled—rather than disabled—by default, single-page Web applications emerged that made heavy use of client-side JavaScript. But there was no good way to deal with client-side storage, databases or even the general state between pages.

Now, however, HTML5 has taken client-side state management head-on and introduced a whole new set of APIs, including specifications such as IndexedDB, Local Storage and the History API, which I'll focus on in this article.

This article is based on the Windows 8 Release Preview. All information is subject to change.

This article discusses:

- The challenge of client-side state management
- Maintaining the URI structure
- The History interface definition
- Using Knockout to help with declarative binding

Technologies discussed:

HTML5 History API, JavaScript, Knockout, Modernizr, Windows 8

Code download available at:

on.csell.net/msdn-historylesson

In short, the goal of the History API is to provide a way for rich JavaScript applications to be better able to not only navigate the session history but also manage its state and provide first-class support for URLs. This means with a few simple API calls and events you can navigate around, pushing and popping state data to the session stack to affect, for example, how the back button operates, while maintaining a great URL structure.

The Problem

When you're working on any solution, it's important to understand the problem. The History API isn't just about having some magic way to persist page state as the end user bangs on the back and forward buttons. While that's certainly true, the real story lies deeper beneath the surface. From a user's perspective, there are two major features of any browser: the URL and the navigation buttons (forward and back). Together these allow the user to request or navigate through a series of documents across the Internet.

Though these features have remained essentially the same over the years, behind the scenes things changed as AJAX caught on. What were once a series of documents became just a single file with AJAX calls behind the scenes. This was great; we could deliver a richer experience with just a little asynchrony, not to mention performance benefits and user experience improvements. But it also posed some problems; it became much more challenging to keep track of the URL, for example, to know where the browser should go when the user hit the back button or refresh.

I don't think I can overstate the importance of the URL. URLs are permanent: users make favorites of them, search engines index

Figure 1 **A Speaker Profile**

```
<PersonViewModel>
  <FirstName>Scott</FirstName>
  <LastName>Hanselman</LastName>
  <Company>Microsoft</Company>
  <Bio>
    My name is Scott Hanselman. I work out of my home office for Microsoft
as a Principal Program Manager, aiming to spread good information about
developing software, usually on the Microsoft stack. Before this I was
the Chief Architect at Corillian Corporation, now a part of Checkfree,
for 6-plus years. I was also involved in a few Microsoft Developer
things for many years like the MVP and RD programs and I'll speak about
computers (and other passions) whenever someone will listen.
  </Bio>
  <Twitter>shanselman</Twitter>
  <WebSite>http://www.hanselman.com</WebSite>
  <Gravatar>/Images/People/2012Speakers/ScottHanselman.png</Gravatar>
</PersonViewModel>
```

them and companies market them. How could you manage them in the changing Web world?

Enter the hash (#) and hashbang (#!). Browsers consider anything after the # as a URL fragment identifier and never send it to the server. The hash was originally meant as a way for anchor tags to link inside the page, but it came to be used for AJAX-related activities. In the early days of AJAX, however, URLs with just a hash wouldn't get indexed by a search engine. The solution was the hashbang, which gave Web applications a way to use and modify a URL and let search engines index it, without ever making a page request back to the server. URLs containing hashbangs, such as twitter.com/#!replies, became important with the advent of Web applications.

The New History API

Though this was all a great step forward, there wasn't really any formal support from the browser. Web applications just played scripting games to make things work. The new History API focuses on giving Web applications the ability to "manage state," which means keeping track of the sequence of documents that make up a session's history. This lets you navigate the session history and persist state data—and deal with the URL properly. To start exploring this API, let's look at the actual interface definition as specified by the World Wide Web Consortium, or W3C (bit.ly/MU89iZ):

```
interface History {
  readonly attribute long length;
  readonly attribute any state;
  void go(optional long delta);
  void back();
  void forward();
  void pushState(any data, DOMString title, optional DOMString url);
  void replaceState(any data, DOMString title, optional DOMString url);
};
```

Not a complicated API, by any stretch.

To get an idea of how things really work, let's start by doing some simple navigation between documents. Assume you have three real documents, a.cshtml, b.cshtml and c.cshtml, and you want to move around in all of them. Typically, you'd simply create an anchor tag goto a that the user clicks on, forcing the browser to drive though its normal page and server lifecycle. As that user clicks around a given Web site, he creates what's called the session history.

There are two potential problems with this method, however. First, you're forcing the browser to drive through its full page cycle, and even call the server; and second, you need a physical document that represents the URL.

AJAX solves part of the problem by allowing you to request just parts of a page, reducing the number of full-page requests and manually updating the URL to something like http://foo.com/#!a or http://foo.com/#a. To accomplish something similar with the History API, you call window.history.pushState(state, title, url), passing along any state you want persisted, along with the title of the page and the URL to be displayed. Note that you're not required to use a URL with a hash or hashbang; you can use just http://foo.com/a, even if "a" doesn't physically exist.

By calling pushState, you're creating the session history for that user's session. The user can navigate as he sees fit and things will work as expected. When he hits the back button, he's taken back to the previous URL as expected, and the URLs that were there before continue to exist, just as with any normal series of pages.

You also have hooks that let you navigate and look around the user's session history. You can dynamically move the user forward and back through the stack, just as if the user himself were clicking the forward and back buttons.

A Real-World Example

Let's make this concrete with a real-world example. I run a technology conference called That Conference (thatconference.com). The conference has many speakers, but I don't want to create a page for each one of them. What I'd prefer to do is dynamically create a page for each speaker that appears real. I can do this easily with the History API.

> I don't think I can overstate the importance of the URL.

As with any script-heavy Web application, I need data. Luckily, That Conference has a simple Representational State Transfer (REST) API I can call to get the speakers, thatConference.com/api/person. This call will yield an array of speakers for the given year in either JSON or XML. **Figure 1** shows an item in that array.

Data is no good without a way to see it. I need to set up a simple markup template I can use to dynamically create a page for each speaker. For this I'm going to use a framework called Knockout (knockoutjs.com). Knockout is a JavaScript library that helps developers use declarative bindings with the Model-View-ViewModel pattern. You're not required to use Knockout for the History API, but I'm going to—and I'll have a little fun along the way.

Because every speaker page is the same, I'm going to define a simple markup template in Knockout. I need to create a script block and tell the framework how to later populate it:

```
<script type="text/html" id="person-template">
  <div>
    <p>
      <strong>Name:</strong>
        <span data-bind="text: FirstName"></span>
        <span data-bind="text: LastName"></span>
    </p>
    <p>Company: <strong data-bind="text: Company"></strong></p>
    <p>Bio: <strong data-bind="text: Bio"></strong></p>
  </div>
</script>
```

Next, I need to populate the template. To do so, I call ko.applyBindings(someData), and Knockout will work its magic on whatever object I pass into applyBindings. With that, I have the basic mechanics in place to take a speaker object and populate the markup with its data.

My goal is a little more complex, though. What I really want is a series of pages that a user can flip through; a book of speakers, if you will. Here's what needs to happen the first time the page is loaded:

1. Get the JSON that represents the speakers.
2. Bind the first item in the array to the Knockout template as the default.
3. Call window.pushState, passing the appropriate arguments.

I've covered the first two steps already, so let's talk about pushState. By calling window.pushState, you're creating an item in the user's session history. I'm going to call pushState and pass three items:

- **State:** In my case this data is the array item I bound to the Knockout template.
- **Title:** This is the title of the page, which will be the speaker's full name.
- **URL:** This is the URL for the page; in this case it will be something like thatconference.com/speakers/speaker/SpeakerFullName.

I've wrapped all of this logic in a method I called bind:

```
function bind (speakerID) {
  var speakerVM = new speakerViewModel(speakerID);
  var fullName = speakers[speakerID].FirstName + speakers[speakerID].LastName
  window.history.pushState(speakerVM, fullName, "/speakers/" + fullName);

  ko.applyBindings(speakerVM);
}
```

Now I'll add a couple of buttons to the speaker book to allow moving through the speakers:

```
<button id="prevSpeaker">previous speaker</button>
<button id="nextSpeaker">next speaker</button>
```

Of course, I need a couple of event handlers for the nextSpeaker and prevSpeaker buttons. To keep track of what speaker should be next, I'm going to create a simple counter that I'll manipulate as the user navigates. The counter value is what I'll pass to the bind method:

```
var counter = 0;

$('#nextSpeaker').click( function () {
  counter = counter + 1;
  bind(counter);
});
$('#prevSpeaker').click( function () {
  counter = counter - 1;
  window.history.back();
});
```

At this point I have a page that loads with some default data, and as I click next, I continue to get the next speaker in the speaker array. If I call prevSpeaker, however, nothing happens. Something more is needed.

Events

When the back button (or a script) calls windows.history.back, the event onpopstate is fired. This is the hook into moving backward in a user's session history. When onpopstate is fired, it passes along the state data given to pushState; in my case, it's that one speaker.

Now I need to grab that state data and tell Knockout to bind it:

```
window.onpopstate = function (event) {
  console.log('onpopstate event was fired');
  ko.applyBindings( event.state );
};
```

With this, I can move back and forth in the session history as expected. You'll now see the speakers change accordingly whether you press the browser's back button or the previous-speaker button.

Now What?

As with most things, the devil is always in the details. I've just scratched the surface of the History API. I didn't cover what to do if a user makes a favorite of a speaker and later visits the site or, for that matter, if he hits refresh while on one of those new speaker pages I just created.

I explicitly didn't cover that scenario because there's a multitude of ways to do so, but they depend on how you've structured your site and the technologies you're using. If you subscribe to using # or #!, calling window.location.hash to get the URL fragment and then calling a service to retrieve the appropriate data for that hash and binding that to your markup might be all you need.

> Rather than basing actions on user agents, you should leverage a tool such as Modernizr to ask the browser what it can do.

It's important to note that while my solution creates an entire dynamic page, you can also use the History API for part of an existing page so the core of the page takes advantage of the server but part of the page uses the History API. You can find a great detailed example of exactly this at bit.ly/vOlB2U.

You should also implement feature detection in your Web application. Rather than basing actions on user agents, you should leverage a tool such as Modernizr (modernizr.com) to ask the browser what it can do. If a user's browser doesn't support a particular feature, you can use a polyfill—a shim that implements that feature for the browser. This can even be done for features such as CSS. For more information about feature detection, check out Brandon Satrom's September 2011 article, "No Browser Left Behind: An HTML5 Adoption Strategy" (msdn.microsoft.com/magazine/hh394148).

AJAX changed the way Web sites interact on the Internet, and Web developers found creative solutions for turning standard Web sites into rich Web applications. The History API is here to help those script-heavy Web applications keep the core browser fundamentals intact.

Everything in this article was done on the Windows 8 Release Preview using Microsoft Web Matrix. You'll find all of the code at on.csell.net/msdn-historylesson and a number of great resources for exploring the History API at on.csell.net/msdn-historylesson-linkstack. ■

CLARK SELL *works as a senior Web and Windows 8 evangelist for Microsoft outside of Chicago. He blogs at csell.net, podcasts at DeveloperSmackdown.com and can be found on Twitter at twitter.com/csell5.*

THANKS to the following technical experts for reviewing this article: *John Hrvatin, Mark Nichols, Tony Ross and Brandon Satrom*

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**

DBAs, IT professionals, developers, and analytics specialists will gather this December in Orlando for comprehensive education, knowledge share and networking—all about and around SQL Server.

Save up to $400!

Register before October 10th

Use Promo Code SQLAUG

Scan the QR code or **visit sqllive360.com** for more information.

Subject-matter experts and industry insiders will guide attendees through mission-critical topics such as:

Friends of SQL Server
Learn to leverage the "friends" of SQL Server – including Integration Services and Reporting Services – more effectively.

Monitoring, Maintaining, and Tuning
Be a real DBA! This track focuses on SQL Server's day-to-day operations – the stuff you need to keep SQL Server running smoothly.

Business Intelligence and Big Data with the Microsoft Stack
Sessions in this track help you understand tie-in technologies to SQL Server and then teach you how to fine-tune them to optimize the solutions you've deployed.

Recovery and High Availability
SQL Server is mission critical – hone your skills to help keep it running and learn new techniques for protecting and recovering data.

What's New in SQL Server 2012
"Denali" is finally here – learn what's new, what's different, and how to get there from whatever version of SQL Server you're currently on.

SQL Server Virtualization and In the Cloud
Virtualize SQL Server? Run SQL Server in "the cloud?" It's all ok as long as you do it right . . . from under-the-hood details to prescriptive advice, this is where to come for the most accurate and up-to-date information in the industry.

GOLD SPONSOR axosoft

SUPPORTED BY Microsoft msdn Visual Studio MAGAZINE Redmond MAGAZINE

MEDIA SPONSOR THE CODE PROJECT

PRODUCED BY 1105 MEDIA

Using the Team Foundation Server Client Object Model

Brian Blackman and Willy-Peter Schaub

In this article we'll introduce the Visual Studio Team Foundation Server (TFS) client object model and create the foundation for a new series of Visual Studio ALM Rangers articles, focused on practical guidance and common coding scenarios with TFS.

To recap, the ALM Rangers are a group of experts who promote collaboration among the Visual Studio product group, Microsoft Services and the Microsoft Most Valuable Professional (MVP) community by addressing missing functionality, removing adoption blockers, and publishing best practices and guidance based on real-world experiences.

Visual Studio TFS is all about application lifecycle management (ALM). It's about enabling teams to collaborate, plan, track, design, develop, build and store their solutions effectively and securely. It's about productivity and bringing ALM within the reach of professional software developers, whether they're from a small, midsize or large organization, and whether they're using a formal, agile or custom process, or working as virtual or tightly coupled teams. In this article we'll explore the basic architecture concepts you need to understand to work with TFS and, more important, to extend its features.

This article discusses:

- The importance of extensibility
- Setting up a dev environment
- Best practices and guidelines
- Namespaces and assemblies
- Connecting to TFS servers

Technologies discussed:

Visual Studio Team Foundation Server

Code download available at:

archive.msdn.microsoft.com/mag201208TFS

> Visual Studio TFS is all about application lifecycle management (ALM).

Why Extensibility Is Important

TFS includes hooks and mechanisms for expanding and enhancing the product with new features. These new features enable you to customize the TFS ALM solution to suit your environment and to extend TFS with custom features and behaviors not built into the standard product.

TFS is a multitier solution, which can be programmatically accessed by using the appropriate object models. As shown in **Figure 1**, the object model includes a client and a server object model, which can be used to extend the client or the application tier, respectively. In this article we'll focus on the client object model.

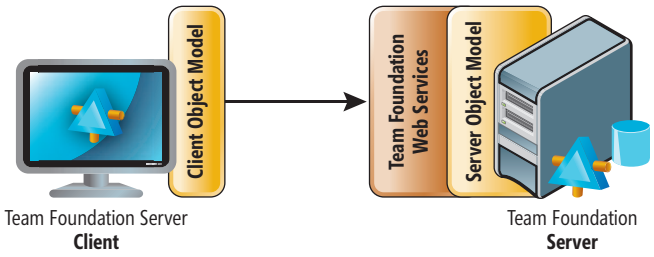


Figure 1 **Visual Studio Team Foundation Server Architecture**

Although it's technically possible to access the Web services or the data tier directly, it's highly recommended to access the TFS features and data through the relevant object models, as shown in **Figure 1**.

Let's briefly take a step back and explore the idea of the TFS object model and its namespaces, which we'll be referring to in both this article and its accompanying sample code download.

The object model is the set of classes logically decomposed into namespaces and physically distributed over a set of assemblies. The TFS client object model is a general term that refers to all public classes available in *Microsoft.TeamFoundation.*.dll* assemblies.

Namespaces organize TFS classes and features into similar items, which could be compared with using a filing cabinet. Each drawer represents an object model. Each folder within each drawer represents a namespace, and the content of each folder represents the classes. Just like a filing cabinet, the namespace allows you to categorize and subsequently find relevant classes easier and more quickly.

We'll focus on getting started and creating a client using the TFS object model to connect to the TFS Web services. The TFS services support specific features such as version control, work item tracking and events.

Setting up a Dev Environment

We'll use the C# language and the Visual Studio IDE. You also can work with the object model with other languages and environments, such as Windows PowerShell.

Use of the TFS client object model currently requires a client SKU—such as Team Explorer—installation on the development machine. In the future, the TFS client object model will be available as a standalone client installation.

Team Explorer is included with all editions of Visual Studio except Express. When you have Visual Studio and TFS installed, start Visual Studio, connect to a TFS server and confirm that you have create, read, update and delete (CRUD) permissions for work items and source control.

For this article and most development work, we used a public virtual machine (VM) provided by Brian Keller from the ALM Developer Evangelism Team at Microsoft (available at bit.ly/VS2010RTMVHD for the Visual Studio 2010 version and aka.ms/VS11ALMVM for the Visual Studio 2012 version). If you download the same VM, the code samples should just work. If you use the code against your own server, it might require some changes.

> Namespaces organize TFS classes and features into similar items, which could be compared with using a filing cabinet.

To work with the sample code in this article, you'll need references to the TFS .NET Framework assemblies listed later. You get these assemblies when you install Team Explorer.

Best Practices and Guidelines

When you develop client applications using the TFS object model, use the following guidelines.

Use the TFS Client API even though the API calls the Web services. Accessing the Web service interfaces directly is not officially supported. The TFS client object model helps with authentication, compatibility with multiple server versions, impersonation, two-factor authentication using client credentials and a handful of other things in addition to providing clean, high-level abstractions over the Web service interfaces. For some clients (for example, a Ruby on Rails app running on *nix), the Web service layer is the only option. If you fall within this category, it's worth checking out the Java SDK (see bit.ly/irHxgm).

Figure 2 **Enabling Tracing in Your Configuration File**

```
<?xml version ="1.0"?>
  <configuration>
    <system.diagnostics>
      <switches>
        <add name="TeamFoundationSoapProxy" value="4" />
        <add name="VersionControl" value="4" />
      </switches>
      <trace autoflush="true" indentsize="3">
        <listeners>
          <add name="rangerListener"
            type="Microsoft.TeamFoundation.TeamFoundationTextWriterTraceListener,
            Microsoft.TeamFoundation.Common, Version=10.0.0.0, Culture=neutral,
            PublicKeyToken=b03f5f7f11d50a3a"
            initializeData="c:\almrangertracing.log" />
          <add name="performanceListener"
            type="Microsoft.TeamFoundation.Client.PerfTraceListener,
            Microsoft.TeamFoundation.Client, Version=10.0.0.0, Culture=neutral,
            PublicKeyToken=b03f5f7f11d50a3a" />
        </listeners>
      </trace>
    </system.diagnostics>
  </configuration>
```

Figure 3 **Web Method Tracing with Performance Data**

```
03/17/2012 12:10:14 (pid 2936, tid 4820, 11636 ms) Web method response:
[http://servername/tfs/myCollection/VersionControl/v1.0/repository.asmx]
QueryWorkspace[VersionControl] 3 ms
03/17/2012 12:10:14 (pid 2936, tid 4820, 11637 ms) CreateWebRequest() -- Uri:
http://servername:8080/tfs/myCollection/VersionControl/v1.0/repository.asmx
03/17/2012 12:10:14 (pid 2936, tid 4820, 11637 ms) request.
AutomaticDecompression: GZip
03/17/2012 12:10:14 (pid 2936, tid 4820, 11637 ms) Web method running:
http:// servername:8080/tfs/myCollection/VersionControl/v1.0/repository.
asmx] UpdateWorkspace[VersionControl]
03/17/2012 12:10:14 (pid 2936, tid 4820, 11658 ms) HTTP headers:
Content-Length: 896
Cache-Control: private, max-age=0
Content-Type: application/soap+xml; charset=utf-8
Date: Sat, 17 Mar 2012 16:10:12 GMT
Server: Microsoft-IIS/7.0
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
```

Figure 4 **TFS SDK Assemblies**

Namespace ☐ = Microsoft.TeamFoundation	Description
☐.client	The namespace and associated assemblies provide the interfaces to connect to TFS and to access data relating to team projects and team project collections.
☐.framework.client	The namespace and associated assemblies expose APIs for viewing and manage the contents of the TFS registry, job schedules, generic property store, event subscriptions, security namespaces, services, team project collections, catalog and Lab Management objects.
☐.framework.common	The namespace and associated assemblies define common objects such as permissions, exceptions and constants.

Where possible, perform recursive operations instead of enumerating each individual item. For example, delete the folder instead of deleting each individual file. This guideline brings up something that we learned from Don Box when we attended his session at TechEd in New Orleans back when he had very long hair and didn't work for Microsoft: "Network traffic is evil." We attended his session on remote procedure calls. Back then, C code was used, but what has not changed is the importance of his statement. Network traffic *is* evil! Some developers haven't yet learned this, or they're learning about it the hard way with customers and consultants complaining about poor performance over the network.

A common pattern will be connecting to Team Foundation Configuration Server or a collection and listing the registered servers or listing the projects.

To adhere to the "network traffic is evil" mantra, try to batch non-recursive operations and only request the download URLs you need when you need them. Some of the accompanying sample code will ignore this guideline in attempts to demonstrate features of the API.

When passing a path to an API, use the server path instead of the local path. Do this because the latter needs to be translated into the server path. This translation requires that the workspace of the caller be identified, the mapping be loaded, the proper mapping be determined and so on, before finally getting the server path.

Use an optimistic pattern with your operations. Just like with databases, assume that you'll be successful at a write operation and catch the exception. For example, in operations that use the version control service, don't spend valuable time checking the item before operating on it, because the file could be deleted, and therefore your need to handle exceptions would be, too. So assume the best and prepare for the worst.

Another useful tool for figuring out where there's evil network traffic is Microsoft Network Monitor. Don't code against the network or debug the network without it. No packet can hide from this tool, and the truth will be revealed. You can download Microsoft Network Monitor from Microsoft downloads (bit.ly/mMmieH).

When you're in a bind and don't know why, or how or for how long things are working, turn on tracing for Web methods. Tracing will show you what Web methods are called by the API. It provides details such as what was called, the duration of the call and the SOAP response. Tracing is turned on through modification of your application .config file in the system.diagnostics section, as shown in **Figure 2**. Sample output that you find in your log file is in **Figure 3**. However, if you want to look at the SOAP payload, you'll need to use a tool such as Microsoft Network Monitor.

Understanding the Assemblies and Namespaces

Many assemblies and namespaces are exposed by the TFS object model. Not all assemblies will be covered in this series of articles, just the most important and most frequently used namespaces. In addition to common TFS namespaces, we'll list .NET core namespaces where the use of the TFS object model has dependencies.

A common pattern will be connecting to Team Foundation Configuration Server or a collection and listing the registered servers or listing the projects. These common client tasks require the following references:

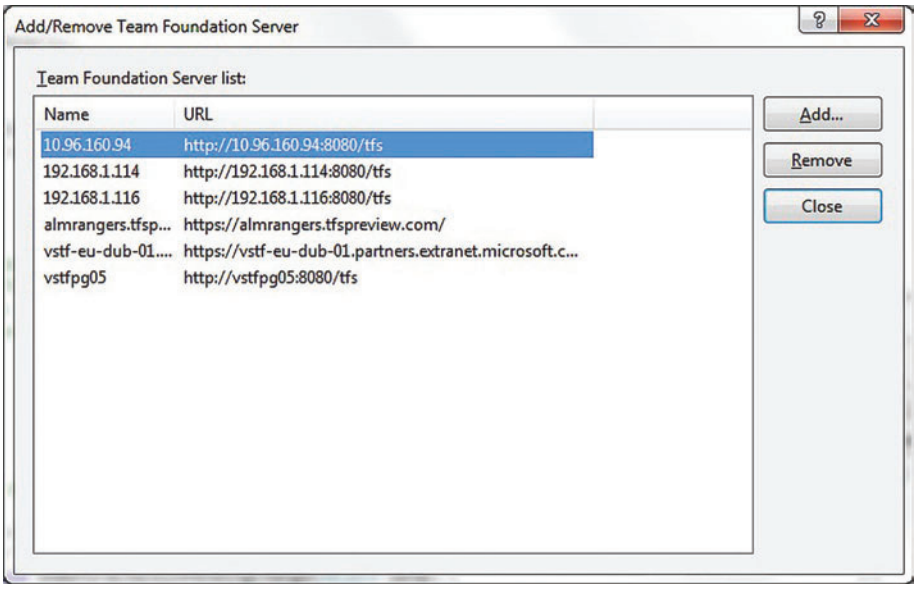


Figure 5 **Registered Team Foundation Server Dialog Box**

```
// Needed for TfsConfigruationServer
using Microsoft.TeamFoundation.Client;

// Needed for Catalog Resources such as types
using Microsoft.TeamFoundation.Framework.Common;

// Needed for other Catalog Resouces such as nodes
using Microsoft.TeamFoundation.Framework.Client;

// Needed for use of the Version Control service
using Microsoft.TeamFoundation.VersionControl.Client;
```

Use of each of the available services will also require the same references, such as Microsoft.TeamFoundation.VersionControl.Client for the Version Control Service. Working with work items could require references to:

- Microsoft.TeamFoundation.WorkItemTracking.Client
- Microsoft.TeamFoundation.WorkItemTracking.Common
- Microsoft.TeamFoundation.WorkItemTracking.Proxy

Figure 4 shows the main assemblies included with the TFS SDK, which are found in Program Files\Microsoft Visual Studio 10.0\Common7\IDE under ReferenceAssemblies\v2.0 and PrivateAssemblies in terms of TFS 2010, or in Program Files (X86)\ ... on a 64-bit OS.

(Note: The default code sample we provided is built and tested with the released TFS 2010 object model, retrieving the referenced assemblies from …\Program Files (x86)\Microsoft Visual Studio 10.0\Common7\IDE\ReferenceAssemblies\v2.0. To use the new TFS 11 beta object model, simply remove the Microsoft.TeamFoundation.* assembly references and re-add them from …Program Files (x86)\Microsoft Visual Studio 11.0\Common7\IDE\ReferenceAssemblies\v2.0 and rebuild the sample applications. You can use either object model to connect to TFS 2010 and 2012 servers using the default samples.)

If you're looking for the complete sample solution and more code samples from the Visual Studio ALM Rangers, or if you'd like to contribute samples yourself, please visit us at bit.ly/MOoyQt.

Our First Programming Adventure

Remember, you'll need the following references to connect to a TFS server:

```
using Microsoft.TeamFoundation.Client;
using Microsoft.TeamFoundation.Framework.Common;
using Microsoft.TeamFoundation.Framework.Client;
```

You could also have a dependency on the System.Net namespace for passing network credentials to some overloaded methods, as shown in the following code:

```
NetworkCredential myNetCredentials =
  new NetworkCredential("Administrator", "P2sswOrd");
ICredentials myCredentials = (ICredentials)myNetCredentials;
```

NetworkCredentials are useful if you're using the public TFS VM. After you have the references, you can connect to a TFS server via several different approaches. You can connect to the configuration server, where you can get a list of collections passing in your server URI and credentials:

```
Uri tfsUri = @"http://servername:8080/tfs";
TfsConfigurationServer configurationServer =
  new TfsConfigurationServer(tfsUri, myCredentials);
```

A better approach is to prompt the user for credentials by passing in UICredentialsProvider, as shown here:

```
TfsConfigurationServer configurationServer =
  new TfsConfigurationServer(tfsUri, new UICredentialsProvider());
```

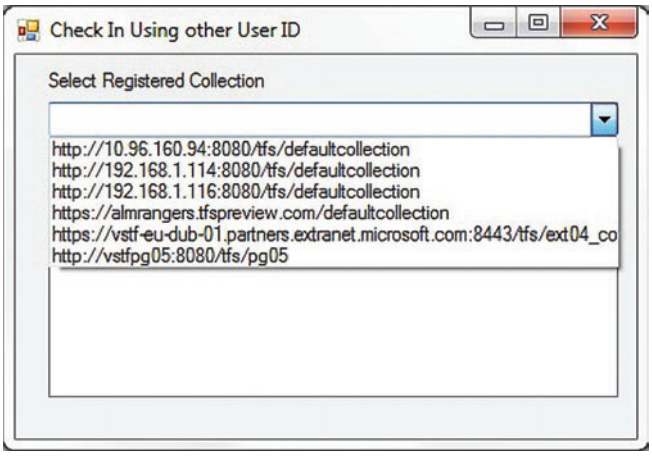


Figure 6 **Reading Registered Servers**

Second Approach—Connecting to a New Configuration Server

Here's an alternate approach when you're running the process under the proper credentials and want to avoid having to connect to a new configuration server:

```
TfsConfigurationServerFactory.GetConfigurationServer(tfsUri);

// Once you have your connection then you can
// get a list of collections on the server
// Get the catalog of team project collections
ReadOnlyCollection<CatalogNode> collectionNodes;
collectionNodes = configurationServer.CatalogNode.QueryChildren(
  new[] { CatalogResourceTypes.ProjectCollection },
  false, CatalogQueryOptions.None);
foreach (CatalogNode collectionNode in collectionNodes)
{
  // Add each collection to the combo box
  cbTPC.Items.Add(collectionNode.Resource.DisplayName);
}
```

The previous example requires that you type a URI in an edit control on the dialog. In the next example, we'll read the list of registered servers on the client. Registered servers are the TFS servers that were added to Visual Studio through its Connect to Team Foundation Server dialog box, as shown in **Figure 5**.

To read the registered servers on the client, you'll need a reference to Microsoft.TeamFoundation.Client. In the following code,

Figure 7 **Connecting to a Known Collection URI**

```
NetworkCredential myNetCredentials =
  new NetworkCredential("Administrator", "P2sswOrd");
ICredentials myCredentials = (ICredentials)myNetCredentials;

// Connect to the tpc hosting the version-control repository
TfsTeamProjectCollection tpc =
  new TfsTeamProjectCollection(new Uri(cb.Text),
  myNetCredentials);

// Get the version control service
vcServer = tpc.GetService<VersionControlServer>();

var teamProjects =
  new List<TeamProject>(vcServer.GetAllTeamProjects(false));

// List the team projects in the collection and add to list box
foreach (TeamProject projectNode in teamProjects)
{
  // Add list or team projects to list box
  lbProjects.Items.Add(projectNode.Name);
}
```

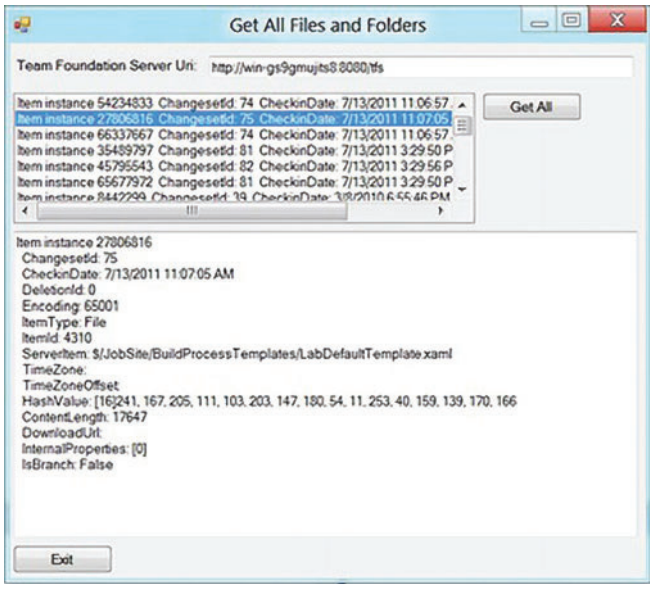


Figure 8 **Get All XAML Files and Show File Details**

the registered servers on the client are read and added to a combo box (shown in **Figure 6**):

```
List<RegisteredProjectCollection> registeredTPCs;
// Get all registered collections on this machine
registeredTPCs = new
  List<RegisteredProjectCollection>((
    RegisteredTfsConnections.GetProjectCollections()));
foreach (var projectCollection in registeredTPCs)
{
  // Add each registered team project collection to the combo box
  cbRegColl.Items.Add(projectCollection.Uri);
}
```

Third Approach—Connecting to a Known Collection URI

In addition to connecting to a server, if you know the URI to the collection, you can connect to it directly. **Figure 7** shows this third approach—connecting to a known collection URI.

The next examples demonstrate using some of the version control service features. When you have the references to the server and a reference to the version control service (see **Figure 7**), you can use some of the APIs, such as getting a list of files on the server or pending changes. The following code uses the version control server reference to ask for all XAML files on the server and adds them to a list box:

```
 // List all of the XAML files on the server
ItemSet items = vcServer.GetItems("$/*.xaml", RecursionType.Full);
foreach (Item item in items.Items)
{
  // Add each file to the list box
  lbAll.Items.Add(item.ToString());
}
```

The first parameter to the GetItems method is the path to the folder. In the sample code, the path is the root, and all XAML files will be returned in all folders. In **Figure 8**, you see that file details are shown when you select one of these files.

If you want to see a list of files with pending changes and add those to a list box, you would use the code in **Figure 9**. In this code, the API QueryPendingSets is overloaded, and this example uses the method overload that allows you to query based on ItemSpecs, which is an array that represents what to query. In this case, we're asking for pending changes for only one folder on the server. We'll also filter on pending changes for a particular workspace and user. The last parameter instructs the API to provide us the information so that we can download the file. If we have no plans to use that information, then consider the mantra "network traffic is evil" and pass false.

> In addition to connecting to a server, if you know the URI to the collection, you can connect to it directly.

The version control service offers many more features that we haven't covered in this article. We tried to whet your appetite and get you started on connecting to the server and using the version control service.

In future articles we'll continue to investigate the TFS client object model and introduce different programming scenarios such as working with work items, version control, builds and deeper integration through the Visual Studio TFS server object model. ■

Figure 9 **Get PendingChanges**

```
ItemSpec[] itemSpecs = new ItemSpec[1];
// You can filter the path to a project and folder
// using $/ProjectName/Folder...
ItemSpecs[0] =
  new ItemSpec(@"$/Tailspin Toys/Development/Iteration 2",
  RecursionType.Full);

// Your options for filtering are the Workspace Name,
// User Name and the previous path in the ItemSpec
String qryWSName = @"WIN-GS9GMUJITS8";
String queryUserName = @"administrator";
bool includeDownloadInfo = true;

PendingSet[] pendingSet = vcServer.QueryPendingSets(
  itemSpecs, qryWSName, qryUserName, includeDownloadInfo);

foreach (PendingSet ps in pendingSet)
{
  // Add pending changes to list box
  lbPendingChanges.Items.Add(ps.ToString());
}
```

Brian Blackman *is a principal consultant with the Microsoft Services Partner ISV team, focusing on affecting ISV partners' success in engineering and the marketplace. He has an MBA and is a CSM, CSP, MCSD (C++), MCTS and Visual Studio ALM Core Ranger. He spends his time writing code, creating and delivering workshops, and consulting in various concentrations and all things ALM.*

Willy-Peter Schaub *is a senior program manager with the Visual Studio ALM Rangers at the Microsoft Canada Development Center. Since the mid-'80s, he's been striving for simplicity and maintainability in software engineering. Read his blog at blogs.msdn.com/b/willy-peter_schaub and follow him on Twitter at twitter.com/wpschaub.*

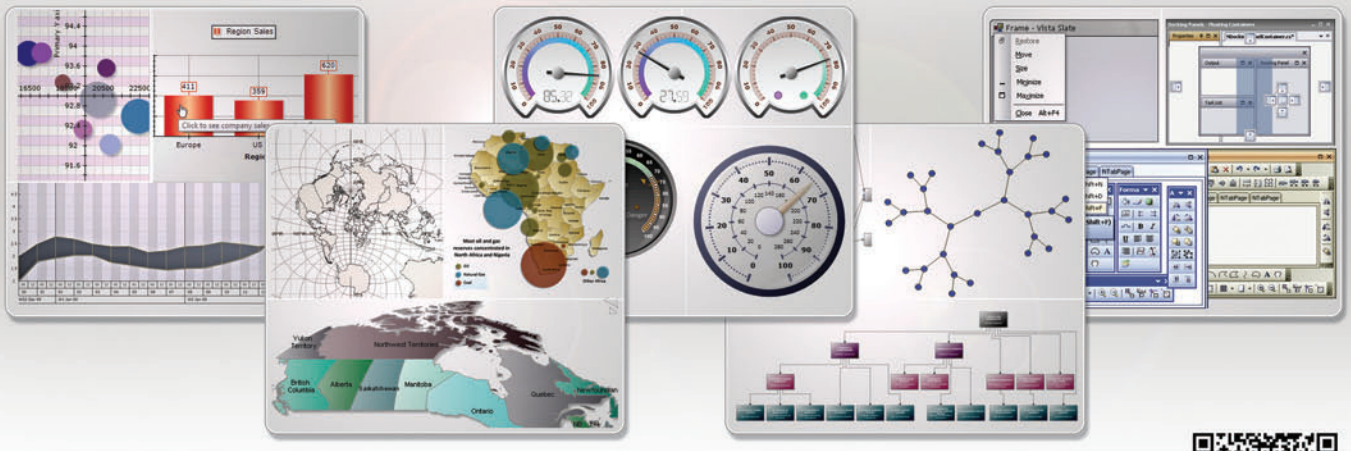
Thanks to the following technical experts for reviewing this article: *Jeff Bramwell, Bill Essary, Mike Fourie, Bijan Javidi, Jim Lamb and Patricia Wagner*

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

CyberNanny: Remote Access via Distributed Components

Angel Hernandez Matos

This article is about an application called CyberNanny, which I recently wrote to allow me to remotely see my baby daughter Miranda at home from anywhere at any time. It's written in Visual C++ (MFC) and it comprises different technologies such as Kinect and its SDK, Windows Azure, Web services and Office automation via Outlook. The project is hosted on CodePlex (cybernanny.codeplex.com), where you can check out the code or contribute to it.

Before I get into the nuts and bolts of the application, I'll briefly explain the technologies used to build it.

C++ has been—and still is—the workhorse in many software shops. Saying that, the new standard C++ 11 takes the language to a new level. Three terms to describe it would be modern, elegant and extremely fast. Also, MFC is still around and Microsoft has been upgrading it with every new release of its Visual C++ compiler.

This article discusses:

- General architecture of the solution
- Kinect architecture
- Locally deployed, native components
- Cloud-hosted, managed components

Technologies discussed:

Windows Azure, Kinect, C++, ASP.NET

Code download available at:

cybernanny.codeplex.com

The Kinect technology is amazing, to say the least; it changes the way we interact with games and computers. And with Microsoft providing developers with an SDK, a new world of opportunities is unveiled for creating software that requires human interaction. Interestingly, though, the Kinect SDK is based on COM (as well as the new programming model in Windows 8, called Windows Runtime, often abbreviated as WinRT). The SDK is also available to Microsoft .NET Framework languages.

One of the requirements I had with CyberNanny was the reliable delivery of messages via a highly available queue, and Windows Azure provides that.

Windows Azure is the Microsoft Platform as a Service (PaaS) offering that has been around for a couple of years. It provides a series of services that allow building solutions on top of them (such as Compute and Storage). One of the requirements I had with CyberNanny was the reliable delivery of messages via a highly available queue, and Windows Azure provides that.

The native use and consumption of Web services is possible using the Windows Web Services API (WWSAPI), which was introduced with Windows 7. I have a blog post (bit.ly/LiygQY) that describes a Windows Presentation Foundation (WPF) application implementing a native component using WWSAPI. It's important to mention that WWSAPI is built in to the OS, so there's no need to download or install anything but the Windows SDK (for header and lib files).

Why reinvent the wheel? One of the requirements for CyberNanny was the ability to send e-mails with attached pictures, so instead of writing my own e-mailing class, I preferred to reuse the functionality provided by Outlook for this task. This allowed me to focus on the main objective: building a distributed application for looking after my baby.

This article is organized in four main sections:

1. Overview of the general architectural solution
2. Kinect architecture
3. Locally deployed components (native)
4. Cloud-hosted components (managed)

Overview of the General Architectural Solution

The CyberNanny concept is simple (as shown in **Figure 1**), but it also has some moving pieces. It can briefly be described as a thick client written in Visual C++, which captures frames via the Kinect sensor. These frames can later be used as a picture that's attached to a new e-mail composed in Outlook through automation. The application is notified about pending requests by spawning a thread triggered from a timer, which polls a queue hosted in Windows Azure. The requests are inserted into the queue via an ASP.NET Web page.

Note that in order to run and test the solution you must have:

- Kinect sensor (I used the one on my Xbox 360)
- Windows Azure subscription
- Kinect SDK

Kinect Architecture

Having a good architectural understanding of how things work and how they can be implemented is crucial to development projects, and in this case Kinect is no exception. Microsoft has provided an SDK for managed and native code developers. I'll describe the architecture Kinect is built upon, as shown in **Figure 2**.

The circled numbers in **Figure 2** correspond to the following:

1. **Kinect hardware:** The hardware components, including the Kinect and the USB hub through which the sensor is connected to the computer.
2. **Kinect drivers:** The Windows drivers for the Kinect, which are installed as part of the SDK setup process as described in this article. The Kinect drivers support:
   - The Kinect microphone array as a kernel-mode audio device that you can access through the standard audio APIs in Windows.
   - Audio and video streaming controls for streaming audio and video (color, depth and skeleton).
   - Device enumeration functions that enable an application to use more than one Kinect.

Having a good architectural understanding of how things work and how they can be implemented is crucial to development projects, and in this case Kinect is no exception.

3. **Audio and video components:** The Kinect Natural User Interface (NUI) for skeleton tracking, audio, color and depth imaging.
4. **DirectX Media Object (DMO):** This is for microphone array beam forming and audio source localization.
5. **Windows 7 standard APIs:** The audio, speech and media APIs in Windows 7, as described in the Windows 7 SDK and the Microsoft Speech SDK.

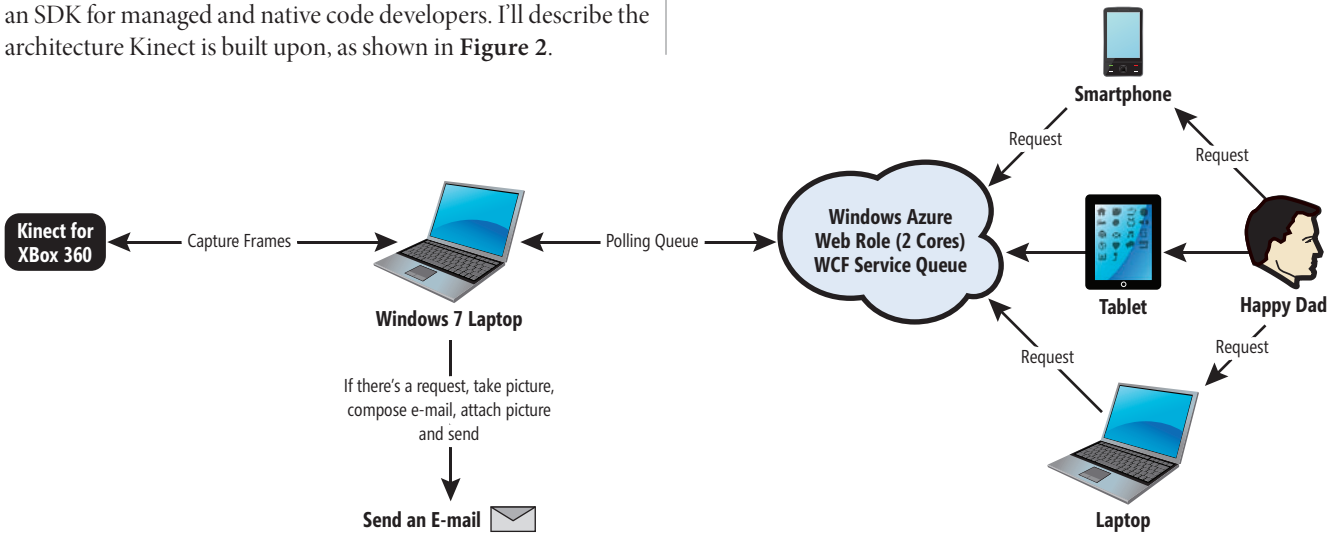


Figure 1 **CyberNanny Architecture**

I'll demonstrate how I used the video component for capturing frames that are then saved as JPEG files for e-mailing purposes. The rendering of the captured frames is done via Direct2D.

The Nui_Core Class I wrote a class called Nui_Core, which encapsulates the functionality I needed from the Kinect sensor. There's a single instance of this object in the application. The application interacts with the sensor via a member of type INuiSensor that represents the physical device connected to the computer. It's important to remember that the Kinect SDK is COM-based, hence the aforementioned interface—as well as all the other COM interfaces used throughout the application—is managed by smart pointers (for example, CComPtr<INuiSensor> m_pSensor;).

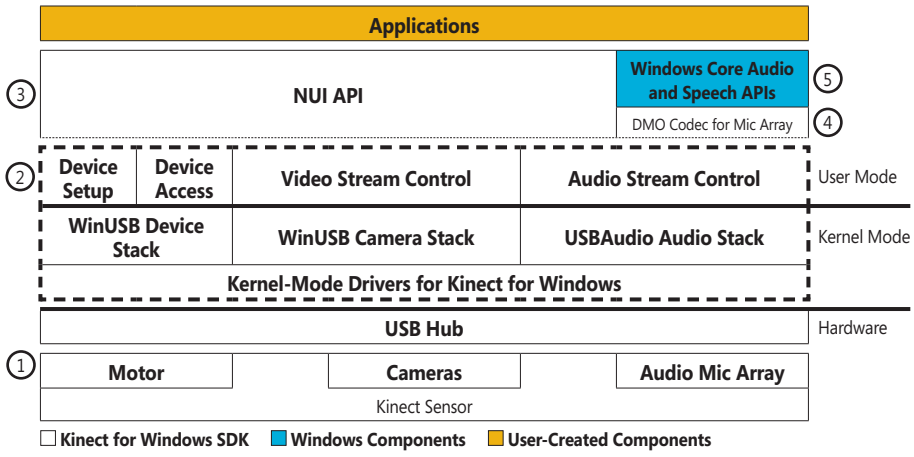


Figure 2 **Kinect for Windows Architecture**

The steps to start capturing frames with the sensor are:

1. Check whether there's a sensor available by calling NuiGetSensorCount.
2. Create an instance of the Kinect sensor by calling NuiCreateSensorByIndex.
3. Create a factory object for the creation of Direct2D resources by calling D2D1CreateFactory.
4. Create events for each stream required by the application.
5. Open the streams by calling NuiImageStreamOpen.
6. Process the captured data (frame).

Once the Nui_Core instance is set up, you can easily take a picture on demand by calling the TakePicture method, as shown in **Figure 3**.

Note that you pass a smart pointer to store the bytes of the image as well as the number of bytes that are copied to it, and then this information is used to handcraft your bitmap.

It's important to mention that once you've finished using the sensor, it has to be shut down by calling NuiShutdown, and handles that were used need to be released.

The DrawDevice Class As previously mentioned, the rendering capabilities are provided by Direct2D; that's why another support class is required for use in conjunction with Nui_Core. This class is responsible for ensuring there are resources available for the captured frame, such as a bitmap in this case.

The three main methods are Initialize, Draw and EnsureResources. I'll describe each.

Initialize: This is responsible for setting up three members of type DrawDevice. The application has a tab control with three tabs, so there's a member for each tab (Color, Skeletal and Depth view). Each tab is a window that's responsible for rendering its corresponding frame. The InitializeColorView shown in the following code is a good example of calling the Initialize method:

```
bool Nui_Core::InitializeColorView() {
  auto width = m_rect.Width();
  auto height = m_rect.Height();
  m_pDrawColor = std::shared_ptr<DrawDevice>(new DrawDevice());
  return (m_pDrawColor.get()->Initialize(m_views[TAB_VIEW_1]->m_hWnd,
  m_pD2DFactory.p, 640, 320, NULL));
}
```

Draw: This renders a frame on the proper tab. It takes as argument a Byte* captured by the sensor. Just as in the movies, the effect of animation comes from the successive rendering of static frames.

EnsureResources: This is responsible for creating a bitmap when requested by the Draw method.

Locally Deployed Components (Native)

The CyberNanny project comprises the following:

- Application
  - CCyberNannyApp (inherited from CWinApp). The application has a single member of type Nui_Core for interacting with the sensor.
- UI Elements
  - CCyberNannyDlg (Main Window, inherited from CDialogEx)
  - CAboutDlg (About Dialog, inherited from CDialogEx)
- Web Service Client
  - Files auto-generated after executing WSUTIL against a service, Web Services Description Language (WSDL). These files contain the messages, structures and methods exposed by the WCF Web service.

Figure 3 **The TakePicture Method**

```
void Nui_Core::TakePicture(std::shared_ptr<BYTE>& imageBytes, int& bytesCount) {
  byte *bytes;
  NUI_IMAGE_FRAME imageFrame;
  NUI_LOCKED_RECT LockedRect;

  if (SUCCEEDED(m_pSensor->NuiImageStreamGetNextFrame(m_hVideoStream,
    m_millisecondsToWait, &imageFrame))) {

    auto pTexture = imageFrame.pFrameTexture;
    pTexture->LockRect(0, &LockedRect, NULL, 0);

    if (LockedRect.Pitch != 0) {
      bytes = static_cast<BYTE *>(LockedRect.pBits);
      m_pDrawColor->Draw(bytes, LockedRect.size);
    }
    pTexture->UnlockRect(0);
    imageBytes.reset(new BYTE[LockedRect.size]);
    memcpy(imageBytes.get(), bytes, LockedRect.size);
    bytesCount = LockedRect.size;
    m_pSensor->NuiImageStreamReleaseFrame(m_hVideoStream, &imageFrame);
  }
}
```

Figure 4 **The ProcessRequest Method Call**

```
void CCyberNannyDlg::ProcessRequest(_request request) {
  if (!request.IsEmpty) {
    auto byteCount = 0;
    ImageFile imageFile;
    std::shared_ptr<BYTE> bytes;
    m_Kinect.TakePicture(bytes, byteCount);
    imageFile.SerializeImage(bytes, byteCount);
    EventLogHelper::LogRequest(request);
    m_emailer.ComposeAndSend(request.EmailRecipient,
    imageFile.ImageFilePath_get());
    imageFile.DeleteFile();
  }
}
```

- Outlook Object Classes
  - In order to manipulate some of the Outlook objects, you have to import them into your project by selecting "Add MFC Class" from ActiveX Control Wizard. The objects used in this solution are Application, Attachment, MailItem and Namespace.
- Proxy
  - This is a custom class that encapsulates the creation of the required objects to interact with WWSAPI.
- Helper Classes
  - These classes are used to support the functionality of the application, such as converting a bitmap into a JPEG to reduce the file size, providing a wrapper to send e-mails and interact with Outlook, and so on.

When the application starts, the following events occur:

1. A new window message is defined by calling RegisterWindowMessage. This is for adding items to the list of events when a request is processed. This is required because you can't directly modify UI elements from a thread different from the UI thread, or you'll incur an illegal cross-thread call. This is managed by the MFC messaging infrastructure.
2. You initialize your Nui_Core member and set up a couple of timers (one for updating the current time on the status bar and another one that kicks off a thread for polling the queue to check whether there's a pending request).
3. The Kinect sensor starts capturing frames, but the application doesn't take a picture unless there's a request in the queue. The ProcessRequest method is responsible for taking a picture, serializing the picture to disk, writing to the event viewer and kicking off the Outlook automation, as shown in **Figure 4**.

The frame originally captured by Kinect is a bitmap that's approximately 1.7MB in size (which isn't convenient for e-mailing and therefore needs to be converted to a JPEG image). It's also upside down, so a 180° rotation is required. This is done by making a couple of calls to GDI+. This functionality is encapsulated in the ImageFile class.

The ImageFile class serves as a wrapper for performing operations with GDI+. The two main methods are:

1. **SerializeImage:** This method takes a shared_ptr<BYTE>, which contains the bytes of the captured frame to be serialized as an image, as well as the count of bytes. The image is also rotated by calling the RotateFlip method.
2. **GetEncoderClsid:** As mentioned, the image file size is too big to use as an attachment—therefore, it needs to be encoded to a format with a smaller footprint (JPEG, for example). GDI+ provides a GetImageEncoders function that lets you find out which encoders are available on the system.

So far I've covered how the application utilizes the Kinect sensor and how the frames captured are used to create a picture for e-mailing. Next, I'll show you how to call the WCF service hosted on Windows Azure.

The frame originally captured by Kinect is a bitmap that's approximately 1.7MB in size (which isn't convenient for e-mailing and therefore needs to be converted to a JPEG image). It's also upside down, so a 180° rotation is required.

WWSAPI, introduced in Windows 7, allows native developers to consume Web or WCF services in an easy and convenient way, without worrying about the communication (sockets) details. The first step for consuming a service is to have a WSDL to use with

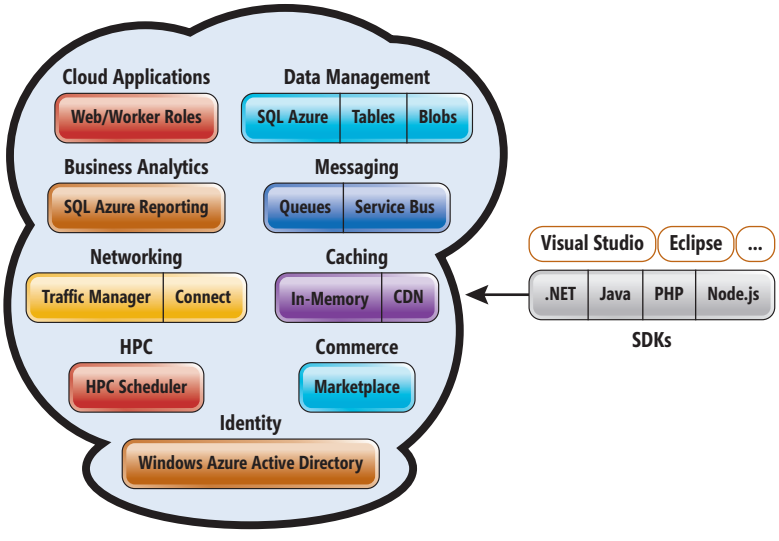


Figure 5 **Windows Azure Platform Services**

WSUTIL that in turn produces codegen C code for service proxies, which are data structures required by the service. There is an alternative called Casablanca (bit.ly/JLletJ), which supports cloud-based client-server communication in native code, but it wasn't available when I wrote CyberNanny.

It's common to get the WSDL and save it to disk, and then use the WSDL file and related schema files as input for WSUTIL. One aspect to take into account is schemas. They must be downloaded along with the WSDL, otherwise WSUTIL will complain when producing the files. You can easily determine the required schemas by checking the .xsd parameter in the schema section of the WSDL file:

```
wsutil /wsdl:cybernanny.wsdl /xsd:cybernanny0.xsd cybernanny1.xsd
cybernanny2.xsd cybernanny3.xsd /string:WS_STRING
```

CyberNanny only has a Web Role that has allocated two cores to guarantee high availability.

The resulting files can be added to the solution, and then you proceed to call your service via the codegen files. Four main objects are required to use with WWSAPI:

1. WS_HEAP
2. WS_ERROR
3. WS_SERVICE_PROXY
4. WS_CHANNEL_PROPERTY

These objects allow the interaction between the client and the service. I put together the functionality to invoke the service in the Proxy class.

Most of the WWSAPI functions return an HRESULT, so debugging errors can be a challenging task. But fear not, because you can enable the tracing from the Windows Event Viewer and see exactly why a given function failed. To enable tracing, navigate to Applications and Services Logs | Microsoft | WebServices | Tracing (right-click it to enable it).

That pretty much covers the native components of the solution. For more information, please refer to the source code on the aforementioned CodePlex site. The next section is about the Windows Azure component of the solution.

Cloud-Hosted Components (Managed)

Please note that this is not an extensive tutorial on Windows Azure, but rather a description of the Windows Azure components in CyberNanny. For more in-depth and detailed information, refer to the Windows Azure Web site at windowsazure.com. The Windows Azure platform (**Figure 5**) comprises the following services:

- Windows Azure Compute
- Windows Azure Storage
- Windows Azure SQL Database
- Windows Azure AppFabric
- Windows Azure Marketplace
- Windows Azure Virtual Network

CyberNanny only has a Web Role that has allocated two cores to guarantee high availability. If one of the nodes fails, the platform will switch to the healthy node. The Web Role is an ASP.NET application, and it only inserts message items into a queue. These messages are then popped out from CyberNanny. There's also a WCF service, which is part of the Web Role that's responsible for handling the queue.

Note that a Windows Azure role is an individual component running in the cloud where each instance of a cloud corresponds to a virtual machine (VM) instance. In CyberNanny's case, then, I've allocated two VMs.

CyberNanny has a Web Role that's a Web application (whether it's only ASPX pages or WCF services) running on IIS. It's accessible via HTTP/HTTPS endpoints. There's also another type of role that's called a Worker Role. It's a background processing application (for example, for financial calculations), and it also has the ability to expose Internet-facing and internal endpoints.

This application also utilizes a queue provided by Windows Azure Storage, which allows reliable storage and delivery of messages. The beauty of the queue is that you don't have to write any specialized code to take advantage of it. Neither are you responsible for setting up the data storage with a certain structure to resemble a queue, because all this functionality is provided out of the box by the platform.

Besides high availability and scalability, one of the benefits provided by the Windows Azure platform is the commonality to do things such as developing, testing and deploying Windows Azure solutions from Visual Studio, as well as having .NET as the lingua franca to build solutions.

This application also utilizes a queue provided by Windows Azure Storage, which allows reliable storage and delivery of messages.

There are some other cool features I'd love to add to CyberNanny, such as motion detection and speech recognition. If you want to use this software or contribute to the project, please feel free to do so. The technologies used are available now and even though they look "different," they can interoperate and play nicely with one another.

Happy coding! ■

Angel Hernandez Matos *is a manager in the Enterprise Applications team at Avanade Australia. He's based in Sydney, Australia, but is originally from Caracas, Venezuela. He has been a Microsoft MVP award recipient for eight consecutive years and is currently an MVP in Visual C++. He has been writing software since he was 12 years old and considers himself an "existential geek."*

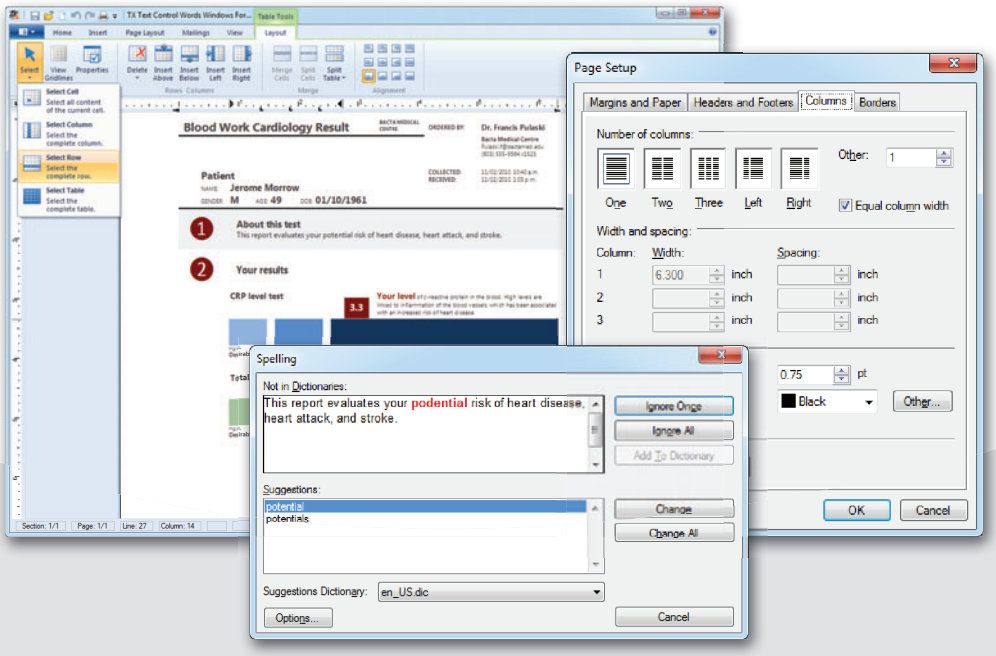
Thanks to the following technical experts for reviewing this article: *Scott Berry, Diego Dagum, Yonghwi Kwon and Nish Sivakumar*

WINDOWS FORMS | WPF | ASP.NET | ACTIVEX

WORD PROCESSING COMPONENTS

Ⓘ Rich Text Editing

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



Rich Text Editing

Add feature-complete word processing capabilities to your Microsoft .NET applications.

Free License

Download our 100% free Express version.

Spell Checking

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

Reporting Redefined

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

TX TEXT CONTROL® word processing components

Microsoft® Visual Studio® PARTNER

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

WWW.TEXTCONTROL.COM

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

.NET Development for ARM Processors

Andrew Pardoe

Consumers are a large driver of the technology market today. As evidenced by the trend known as "the consumerization of IT," long battery life and always-connected and media-rich experiences are important to all technology customers. To enable the best experience for devices with long battery life, Microsoft is bringing the Windows 8 OS to systems built on the low-powered ARM processor, which powers most mobile devices today. In this article, I'll discuss details about the Microsoft .NET Framework and the ARM processor, what you as a .NET developer should keep in mind and what we at Microsoft (where I'm a program manager on the CLR team) had to do to bring .NET over to ARM.

As a .NET developer, you can imagine that writing apps to run on a variety of different processors would pose a bit of a quandary. The ARM processor's instruction set architecture (ISA) is incompatible with the x86 processor's ISA. Apps built to run natively on x86 run well on x64 because the x64 processor's ISA is a superset of the x86 ISA. But the same isn't true of native x86 apps running on ARM—they need to be recompiled to execute on the incompatible architecture. Being able to choose from a range of different devices is great for consumers, but it brings some complexity to the developer story.

Writing your app in a .NET language not only allows you to reuse your existing skills and code, it also enables your app to run on all Windows 8 processors without recompilation. By porting the .NET Framework to ARM, we helped to abstract the unique characteristics of the architecture that are unfamiliar to most Windows developers. But there are still some things you might need to watch out for when writing code to run on ARM.

This article discusses:

- ARM processors and different .NET versions
- Considerations for .NET developers targeting ARM
- Technical details about ARM support in .NET

Technologies discussed:

Microsoft .NET Framework, ARM processors

The Road to ARM: .NET Past and Present

The .NET Framework already runs on ARM processors, but it's not the exact same .NET Framework as the version that runs on the desktop. Back when we started working on the first version of .NET, we realized that being able to write easily portable code across processors was key to our value proposition of increased developer productivity. The x86 processor dominated the desktop computing space, but a huge variety of processors existed in the embedded and mobile space. To enable developers to target those processors, we created a version of the .NET Framework called the .NET Compact Framework, which runs on machines that have memory and processor constraints.

The first devices that the .NET Compact Framework supported had as little as 4MB of RAM and a 33 MHz CPU. The design of the

.NET Compact Framework emphasized both an efficient implementation (which allowed it to run on such constrained devices) and portability (so it could run across the vast range of processors that were common in the mobile and embedded spaces). But the most popular mobile devices—smartphones—now run on configurations that are comparable to the computers of 10 years ago. The desktop .NET Framework was designed to run on Windows XP machines with at least a 300 MHz processor and 128MB of RAM. Windows Phone devices today require at least 256MB of RAM and a modern ARM Cortex processor.

The .NET Compact Framework is still a big part of the Windows Embedded Compact developer story. Devices in embedded scenarios run in constrained configurations, often with as little as 32MB of RAM. We've also created a version of the .NET Framework called the .NET Micro Framework, which runs on processors that have as little as 64KB of RAM. So we actually have three versions of the .NET Framework, each of which runs on a different class of processor. But this is the first time that our flagship product, the desktop .NET Framework, has joined the Compact and Micro Frameworks in running on ARM processors.

Running on ARM

Although the .NET Framework was designed to be platform-neutral, it's mostly been run on x86-based hardware throughout its existence. This means that a few x86-specific patterns have slipped into the collective minds of .NET programmers. You should be able to focus on writing great apps instead of writing for the processor architecture, but you should keep a few things in mind when writing .NET code to run on ARM. These include a weaker memory model and stricter data alignment requirements as well as some places where function parameters are treated differently. Finally, there are a few project-configuration steps in Visual Studio that differ when you target devices. I'll discuss each of these.

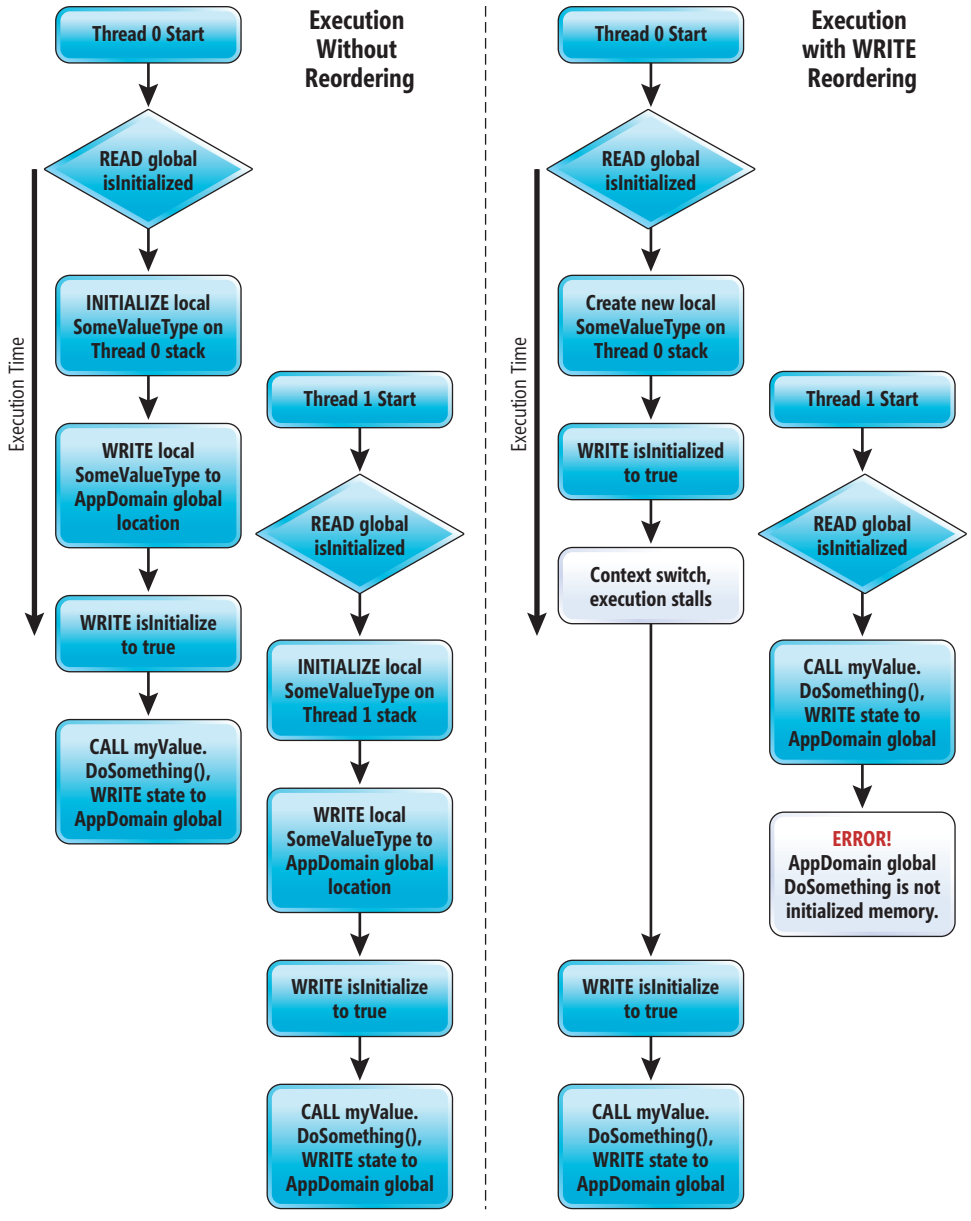


Figure 1 **Write Reordering**

A Weaker Memory Model A "memory model" refers to the visibility of changes made to global state in a multithreaded program. A program that shares data between two (or more) threads will normally take a lock on that shared data. Depending on the particular lock used, if one thread is accessing the data, other threads that attempt to access the data will block until the first thread is finished with the shared data. But locks aren't necessary if you know that every thread accessing the shared data will do so without interfering with other threads' view of that data. Programming in such a manner is called using a "lock-free" algorithm.

The trouble with lock-free algorithms comes when you don't know the precise order in which your code will execute. Modern processors reorder instructions to ensure the processor can make progress on every clock cycle and combine writes to memory in order to decrease latency. Although almost every processor performs these optimizations, there's a difference in how the ordering of reads and writes is presented to

the program. x86-based processors guarantee that the processor will look like it's executing most reads and writes in the same order that the program specifies them. This guarantee is called a strong memory model, or strong write ordering. ARM processors don't make as many guarantees—they're generally free to move operations around as long as doing so doesn't change the way the code would run in a single-threaded program. The ARM processor does make some guarantees that allow carefully constructed lock-free code, but it has what's called a weak memory model.

Interestingly, the .NET Framework CLR itself has a weak memory model. All the references to write ordering in the ECMA Common Language Infrastructure (CLI) specification (available as a PDF at bit.ly/1Hv1xw), the standard that the CLR is designed to meet, refer to volatile accesses. In C# this means accesses to variables marked with the *volatile* keyword (see section 12.6 of the CLI specification for reference). But in the last decade, most managed code has been run on x86 systems, and the CLR just-in-time (JIT) compiler hasn't added much to the reorderings permitted by the hardware, so there were relatively few cases where the memory model would reveal latent concurrency bugs. This could present a problem if managed code written for and tested only on x86-based machines is expected to work the same way on ARM systems.

Most of the patterns that require additional caution with regard to reordering are rare in managed code. But some of these patterns that do exist are deceptively simple. Here's an example of code that doesn't look like it has a bug, but if these statics are changed on another thread, this code might break on a machine with a weak memory model:

```
static bool isInitialized = false;
static SomeValueType myValue;
if (!isInitialized)
{
  myValue = new SomeValueType();
  isInitialized = true;
}
myValue.DoSomething();
```

To make this code correct, simply indicate that the isInitialized flag is volatile:

```
static volatile bool isInitialized = false; // Properly marked as volatile
```

Execution of this code without reordering is shown in the left block in **Figure 1**. Thread 0 is the first to initialize SomeValueType on its local stack and copies the locally created SomeValueType to an AppDomain global location. Thread 1 determines by checking isInitialized that it also needs to create SomeValueType. But there's no problem because the data is being written back to the same AppDomain global location. (Most often, as in this example, any mutations made by the DoSomething method are idempotent.)

The right-side block shows execution of the same code with a system that supports write reordering (and a conveniently placed stall in execution). This code fails to execute properly because Thread 1 determines by reading isInitialized's value that SomeValueType doesn't need to be initialized. The call to DoSomething refers to memory that hasn't been initialized. Any data read from SomeValueType will have been set by the CLR to 0.

Your code won't often execute incorrectly due to this kind of reordering, but it does happen. These reorderings are perfectly legal—the order of the writes doesn't matter when executing on a single thread. It's best when writing concurrent code to mark volatile variables correctly with the volatile keyword.

The CLR is allowed to expose a stronger memory model than the ECMA CLI specification requires. On x86, for example, the memory model of the CLR is strong because the processor's memory model is strong. The .NET team could've made the memory model on ARM as strong as the model on x86, but ensuring the perfect ordering whenever possible can have a notable impact on code execution performance. We've done targeted work to strengthen the memory model on ARM—specifically, we've inserted memory barriers at key points when writing to the managed heap to guarantee type safety—but we've made sure to only do this with a minimal impact on performance. The team went through multiple design reviews with experts to make sure that the techniques applied in the ARM CLR were correct. Moreover, performance benchmarks show that .NET code execution performance scales the same as native C++ code when compared across x86, x64 and ARM.

If your code relies on lock-free algorithms that depend on the implementation of the x86 CLR (rather than the ECMA CLR specification), you'll want to add the *volatile* keyword to relevant variables as appropriate. Once you've marked shared state as volatile, the CLR will take care of everything for you. If you're like most developers, you're ready to run on ARM because you've already used locks to protect your shared data, properly marked volatile variables and tested your app on ARM.

Data Alignment Requirements Another difference that might affect some programs is that ARM processors require some data to be aligned. The specific pattern in which alignment requirements apply is when you have a 64-bit value (that is, an int64, a uint64 or a double) that isn't aligned on a 64-bit boundary. The CLR takes care of alignment for you, but there are two ways to force an unaligned data type. The first way is to explicitly specify the layout of a structure with the [ExplicitLayout] custom attribute. The second way is to incorrectly specify the layout of a structure passed between managed and native code.

If you notice a P/Invoke call come back with garbage, you might want to take a look at any structures being marshaled. As an example, we fixed a bug while porting some .NET libraries in which a COM interface passed a POINTL structure containing two 32-bit fields to a function in managed code that took a 64-bit double as a parameter. The function used bit operations to obtain the two 32-bit fields. Here's a simplified version of the buggy function:

```
void CalledFromNative(int parameter, long point)
{
  // Unpack native POINTL from long point
  int x = (int)(point & 0xFFFFFFFF);
  int y = (int)((point >> 32) & 0xFFFFFFFF);
  ...  // Do something with POINTL here
}
```

The native code didn't have to align the POINTL structure on a 64-bit boundary, because it contained two 32-bit fields. But ARM requires the 64-bit double to be aligned when it's passed into the managed function. Making certain that the types are specified to be the same on both sides of the managed-native call is critical if your types require alignment.

Inside Visual Studio Most developers won't ever notice the differences I've discussed because .NET code by design isn't specific to any processor architecture. But there are some differences in Visual Studio when profiling or debugging Windows 8 apps on

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.

- **Visual Studio / .NET**
- **HTML5**
- **Cloud Computing and Services**
- **Windows 8 / WinRT**
- **Data Management**
- **Silverlight / WPF**
- **Windows Phone**

Register Today and Save $400!

Use Promo Code AUG1AD

SUPPORTED BY

Microsoft | Visual Studio | msdn | Visual Studio MAGAZINE

PRODUCED BY

1105 MEDIA

MEDIA SPONSOR

THE CODE PROJECT WWW.CODEPROJECT.COM

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

an ARM device, because Visual Studio doesn't run on ARM devices.

You're already familiar with the cross-platform development process if you write apps for Windows Phone. Visual Studio runs on your x86 dev box and you launch your app remotely on the device or an emulator. The app uses a proxy installed on your device to communicate with your development machine through an IP connection. Other than the initial setup steps, the debugging and profiling experiences behave the same on all processors.

One other point to be aware of is the Visual Studio project settings add ARM to x86 and x64 as a choice of target processor. You'll normally choose to target AnyCPU when you write a .NET app for Windows on ARM, and your app will just run on all Windows 8 architectures.

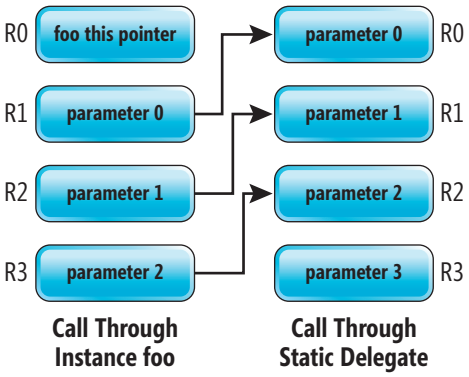


Figure 2 **A "Shuffle Thunk" Shuffles Value Parameters Across Registers**

Going Deep into Supporting ARM

Having made it this far into the article, you already know a lot about ARM. Now I'd like to share some of the interesting, deep technical details about the .NET team's work to support ARM. You won't have to do this kind of work in your .NET code—this is just a quick behind-the-scenes peek at the kind of work we did.

Most of the changes inside the CLR itself were straightforward because the CLR was designed to be portable across architectures. We did have to make a few changes to conform to the ARM Application Binary Interface (ABI). We also had to rewrite assembly code in the CLR to target ARM and change our JIT compiler to emit ARM Thumb 2 instructions.

The ABI specifies the *how* of a processor's programmable interface. It's similar to the API that specifies the *what* of an OS's programmatically available functions. The three areas of the ABI that affected our work are the function calling convention, the register conventions and call stack unwind information. I'll discuss each.

Function Calling Convention A calling convention is an agreement between code that calls a function and the function being called. The convention specifies how parameters and return values are laid out in memory, as well as what registers need to have their values preserved across the call. In order for function calls to work across boundaries (such as a call from a program into the OS), code generators need to generate function calls that match the convention that the processor defines, including alignment of 64-bit values.

ARM was the first 32-bit processor where the CLR had to handle aligning parameters and objects on the managed heap on a 64-bit boundary. The simplest solution would be to align all parameters, but the ABI requires that a code generator not leave bubbles in the stack when no alignment is required so there's no performance degradation. Thus the simple operation of pushing a bunch of parameters on the stack becomes more delicate on the ARM processor. Because a user structure can contain an int64, the CLR's solution was to use a bit on each type to indicate if it requires alignment. This gives the CLR enough information to ensure that function calls containing 64-bit values don't accidentally corrupt the call stack.

Register Convention The data-alignment requirement carries over when structures are fully or partially registered on ARM. This means we had to modify the code inside the CLR that moves frequently used data from memory into registers to make sure the data is properly aligned in the registers. This work had to be done for two situations: first, making certain that 64-bit values start in even registers, and second, placing homogeneous floating-point aggregates (HFAs) in the proper registers.

If a code generator registers an int64 on ARM, it must be stored in an even-odd pair—that is, R0-R1 or R2-R3. The protocol for HFAs allows up to four double or single floating-point values in a homogeneous structure. If these are registered, they must be stored in either the S (single) or D (double) register sets but not in the general-purpose R registers.

Unwind Information Unwind information records the effects that a function call has on the stack and records where nonvolatile registers are saved over function calls. On x86, Windows looks at FS:0 to view a linked list of each function's exception registration information in the event of an unhandled exception. 64-bit Windows introduced the concept of unwind information that allows Windows to crawl the stack in case of an unhandled exception. The ARM design extended this unwind information from 64-bit designs. The CLR code generators, in turn, had to change to accommodate the new design.

> The most significant issues you'll see when porting your app to ARM are performance differences from desktop processors.

Assembly Code Even though most of the CLR runtime engine is written in C++, we have assembly code that must be ported to each new processor. Most of this assembly code is what we call "stub functions," or "stubs." Stubs serve as interface glue that enables us to tie together the C++ and JIT-compiled portions of the runtime. The remainder of the assembly code inside the CLR is written in assembly for performance. For example, the garbage collector write barrier needs to be extremely fast because it's called frequently—any time an object reference is written to an object on the managed heap.

One example of a stub is what we call the "shuffle thunk." We call it a shuffle thunk because it shuffles parameter values across registers. Sometimes the CLR has to change the placement of

parameters in registers just before a function call is made. The CLR uses the shuffle thunk to do this when invoking delegates.

Conceptually, when you invoke a delegate, you just make a call to the Invoke method. In reality, the CLR makes an indirect call through a field of the delegate rather than making a named method call (except when you explicitly call Invoke through reflection). This method is far faster than a named method call because the runtime can simply swap the instance of the delegate (obtained from the target pointer) for the delegate in the function call. That is, for an instance foo of delegate d, the call to the d.Member method is mapped to the foo.Member method.

If you make a closed instance delegate call, the *this* pointer is stored inside the first register used for passing parameters, R0, and the first parameter is stored in the next register, R1. But this only works when you have a delegate bound to an instance method. What happens if you're calling an open static delegate? In that case, you expect that the first parameter is stored in R0 (as there's no *this* pointer.) The shuffle thunk moves the first parameter from R1 into R0, the second parameter into R0 and so on, as show in **Figure 2**. Because the purpose of this shuffle thunk is to move values from register to register, it needs to be rewritten specifically for each processor.

Just Focus on the Code

To review, porting the .NET Framework to ARM was an interesting project and a lot of fun for the .NET team. And writing .NET apps to run on top of the .NET Framework on ARM should be fun for you as a .NET developer. Your .NET code might execute differently on ARM than it does on x86-based processors in a few situations, but the .NET Framework virtual execution environment normally abstracts those differences for you. This means you don't have to worry about what processor architecture your .NET app runs on. You can just focus on writing great code.

I believe having Windows 8 available on ARM will be great for both developers and end users. ARM processors are especially suited for long battery life, so they enable lightweight, portable, always-connected devices. The most significant issues you'll see when porting your app to ARM are performance differences from desktop processors. But make sure to run your code on ARM before saying it actually works on ARM—don't trust that developing on x86 is enough. For most developers, that's all that's needed. And if you do run into any issues, you can refer back to this article to get some insight into where to start investigating. ■

ANDREW PARDOE *is a program manager for the CLR team, helping to ship the Microsoft .NET Framework on all kinds of processors. His personal favorite remains the Itanium. He can be reached at Andrew.Pardoe@microsoft.com.*

THANKS to the following technical experts for reviewing this article: *Brandon Bray, Layla Driscoll, Eric Eilebrecht and Rudi Martin*

TED NEWARD

Cassandra NoSQL Database: Getting Started

The ancient Greeks told the story of Cassandra, the daughter of King Priam and Queen Hecuba of Troy. She was one of the most beautiful women of her generation. When offered the gifts of a prophetess by the Greek god Apollo, she quickly accepted, but when she later spurned his amorous advances, Apollo cursed her to always know the truth and never be believed by any to whom she spoke it. Thanks to her gift of prophesy, Cassandra foresaw the trap presented by the Trojan horse, but thanks to her curse of disbelief, no one in Troy would listen to her warnings. They brought the horse within the city walls, and unwittingly invited the Greek soldiers hidden therein into the city, which led to Troy's fall. Cassandra was taken as a war prize back to Greece by Agamemnon, where she again foresaw the future: his (and her) death, but was again disbelieved—and, sure enough, both he and she were killed.

Modern computer science geeks tell the story of Cassandra a little differently, as Apache Cassandra, another of the "NoSQL" databases—and a popular one at that—in use at a variety of well-known Internet-based companies (YouTube, Netflix and others), and presumably one whose reports are actually taken at face value. (Rumor has it that Cassandra is a pun on another famous prophetess, the Oracle of Delphi.)

To the developer, Cassandra the software can be just as confusing as Cassandra the Trojan. It's "an open source, distributed, decentralized, elastically scalable, highly available, fault-tolerant, tuneably consistent, column-oriented database that bases its distribution design on Amazon's Dynamo and its data model on Google's Bigtable" (source: "Cassandra: The Definitive Guide," O'Reilly Media, 2010, p. 14).

Sometimes I think the Greek myths make more sense than my industry does.

Breaking all that down, we see that:

- Cassandra is built to store lots and lots and lots of data (hundreds of terabytes seem to be a commonly cited example) across a variety of machines arranged in a ring, as opposed to the trend within relational database thinking that says "buy a bigger box" (for scaling horizontally, rather than vertically).

To the developer, Cassandra the software can be just as confusing as Cassandra the Trojan.

- Cassandra has a data model that looks like the relational database's data model on the surface, sounds kind of like it with its discussions of columns, column families and named values, but acts nothing like it in practice.

More relevant to this discussion, Cassandra has been gaining momentum within the developer community as a worthwhile tool to have in the toolbox, so it seemed like a good idea to turn our collective columnar gaze upon a column-oriented database. (Pun intended.)

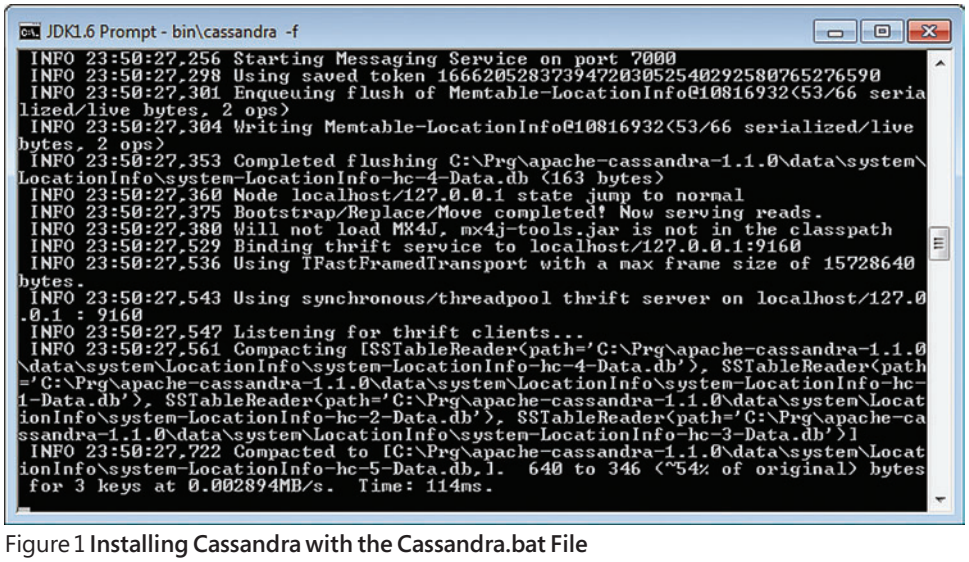


Figure 1 **Installing Cassandra with the Cassandra.bat File**

Conceptual Overview

Cassandra is not a relational data store, despite its use of the term "column-oriented." In fact, it doesn't really look anything at all like a relational database. Instead of storing a schema, for example, that guarantees the various rows of data in the table are all alike, Cassandra stores "column families" in "keyspaces." A keyspace is really

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

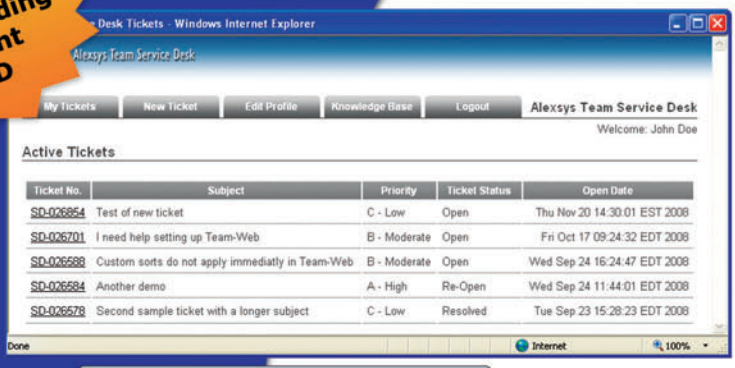
Service Desk Features

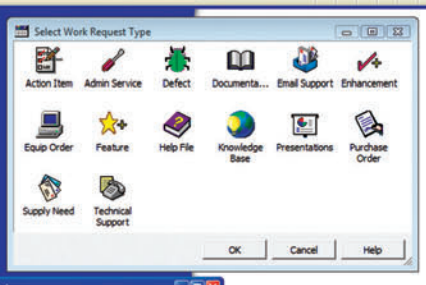
- Fully Secure
- Unlimited Users Self Registered or Active Directory
- Integrated into Your Web Site
- Fast/AJAX Dynamic Content
- Unlimited Service Desks
- Visual Service Desk Builder

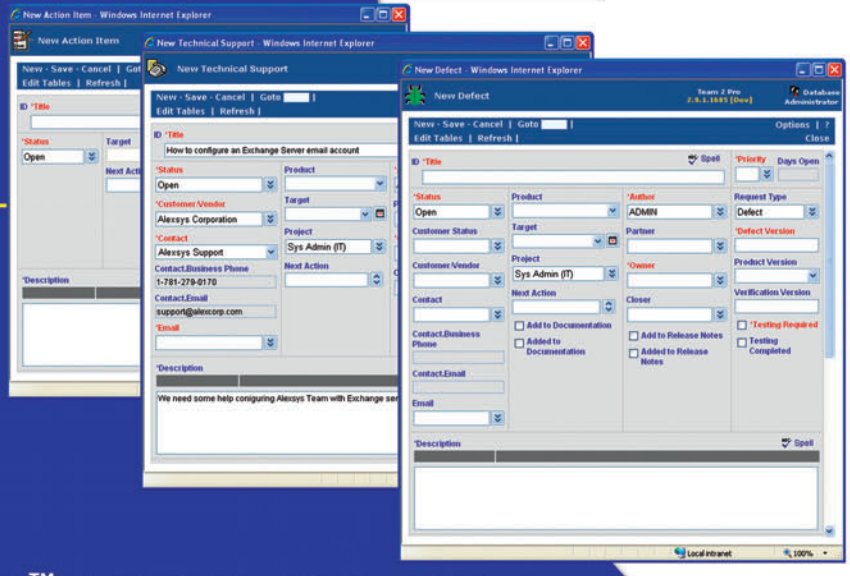
Team 2 Features

- 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.

just an administrative isolation barrier, in much the same way that relational database instances are separated from one another on the same server, but a column family is a completely different beast. Each column family is made up of "rows" identified by a key, but within a row, any number of name/value pairs (columns) can be present, and each row can contain entirely different data elements from the other rows within the column family.

In practical terms, let's suppose we're using Cassandra to store a collection of people. Within the keystore "Earth," we'll have a column family called "People," which in turn has rows that 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 (though if the variance grows too large, it might get confusing for developers to use). Storing pets in here, for example, would likely create too much chaos. This is why any nontrivial application will likely use dozens or hundreds of different column families.

By the way, I'm lying (slightly) to you when I say that a row is made up of name/value pairs; it's actually made up of name/value/timestamp triplets, but the Cassandra docs make it pretty clear that the timestamp part of the triplet is only for conflict detection and is never to be used as part of your application logic. Most Cassandra articles essentially tell new Cassandra developers to ignore it.

This all makes more sense once you see it in action, so let's get Cassandra running.

Getting Started

Before you can do anything with Cassandra, you have to get it installed, and therein lies the first hurdle: Cassandra is, as advertised, an open source project, and like many open source projects, it's not written in a Microsoft .NET Framework language. Instead, Cassandra is written in Java, and as such requires a relatively modern Java runtime to be installed on your machine in order to execute. Cassandra runs fine with Java 6 (and, in fact, most of the blog posts on the subject suggest it), but should run just as well if not a touch faster with the most recently released Java 7.

(If you've never installed Java on your machine before, just plug "Java Runtime Environment 6 (or 7) download" into your search engine of choice and pull down the desired installer for either 32- or 64-bit Windows, depending on your target OS. About the only other thing you'll need to do is set an environment variable called JAVA_HOME to point to the Java Runtime Environment (JRE) install directory—under a default installation, this will be in C:\Program Files\Java\jre6—and put the JRE's "bin" subdirectory on the PATH if it's not already.)

> Cassandra has been gaining momentum within the developer community as a worthwhile tool to have in the toolbox.

Next, pull down the Cassandra binaries from the Cassandra homepage. Unfortunately for us Windows folks, it's only available as a .tar.gz file, which, out of the box, Windows isn't sure what to do with. Dozens of tools are available to unarchive a .tar.gz file, including the command-line "gunzip" and "tar" utilities in Cygwin, if you want to start practicing some Unix-Fu on a Windows box. Dump the contents of the Cassandra download into a convenient directory, such as C:\Prg\apache-cassandra-1.1.0 (which is the latest version, as I write this). Then, as is common with Java projects, you need to create an environment variable that points to the root of the Cassandra install directory, so create a CASSANDRA_HOME environment variable that points to C:\Prg\apache-cassandra-1.1.0 (in my case).

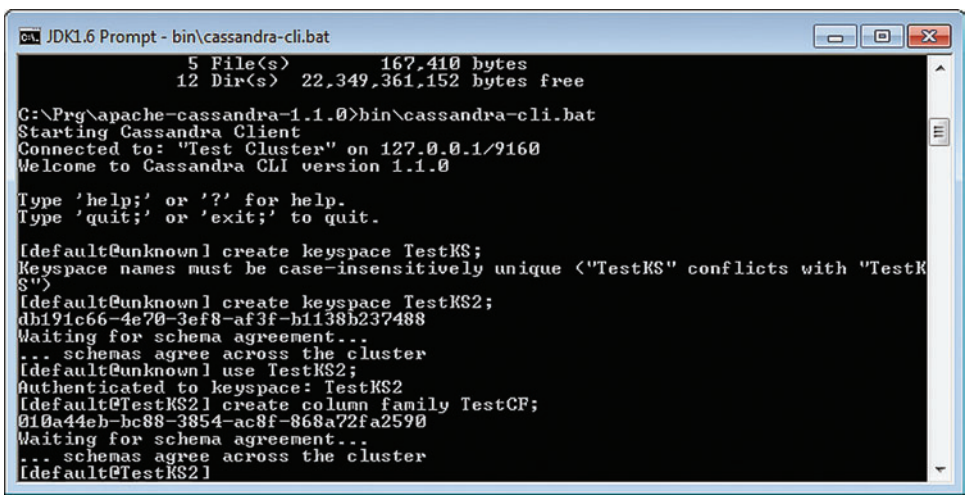


Figure 2 **Connecting to a Running Cassandra Instance**

If you're a little aghast at the primitive conditions here, remember that Java projects like to work on multiple platforms (which means we have to use mechanisms that are common to all platforms, and yeah, environment variables are everywhere, even on Android). The positive side of this is that if you ever work with Cassandra on a non-Windows platform, you'll be doing the same setup steps: get Java, get Cassandra, unarchive and set environment variables. Unfortunately, it means that our tooling isn't quite as fancy and GUI-based as we might otherwise be used to.

Speak to Us, O Prophetess!

Speaking of which, firing up Cassandra means hopping on over to the Cassandra install directory and kicking off the batch file "cassandra.bat" found in the "bin" subdirectory. Launch that as "cassandra –f" (the "-f" causes it to run in the foreground), and you should see something like **Figure 1**.

By default, Cassandra is configured to dump data and commit logs into the "var" directory off the root of your filesystem, which Java interprets as C:\. This is more Unix-ism, and is easily configured differently in the "conf/cassandra.yaml" configuration file.

> Cassandra is written in Java, and as such requires a relatively modern Java runtime to be installed on your machine in order to execute.

(Convenience note: A company called DataStax Inc. offers an all-in-one installer containing both the Cassandra server and JRE, as well as an HTML-based operation center product, available as a free download. If you're having difficulties getting it all set up, you might try that instead.)

A running Cassandra server is expecting incoming connections on port 9160 and uses port 7199 for its Java Management Extensions monitoring, which is Java's rough equivalent to Windows Management Instrumentation. Both ports will, eventually, want to be accessible to client applications and Cassandra monitoring utilities, respectively.

Once Cassandra is up and running on your box, we can connect to the running instance using the Cassandra command-line interface, launched by running "cassandra-cli.bat," again from the Cassandra "bin" directory (see **Figure 2**).

To create a keyspace, use "create keyspace TestKS" (which must be a unique name), and to create a column family within that keyspace, first type "use <keyspace>," then "create column family <name>." No other schema definition is required—the column family is a collection of name/value pairs from then on, remember.

To insert data into the column family, use the "set" command, which requires the name of the column family into which you insert ("TestCF"), the key to use for this row ("TestKey"), the column within the column family to use as the name for this value ("column") and the value to store there ("value"). However, because Cassandra stores data as binary values, you have to tell Cassandra to interpret the row key, column name and column value as ASCII values using the built-in "ascii" function. This means the whole "set" looks like this:

```
set TestCF[ascii('TestKey')][ascii('column')]=ascii('value');
```

Retrieving that data is basically the same exercise using the "get" command, like this:

```
get TestCF[ascii("TestKey")];
```

This will return with something like this:

```
(column=636f6c756d6e, value=76616c7565, timestamp=1338798419726000)
```

This demonstrates that Cassandra does, indeed, speak gibberish (at least, to us humans—if you look carefully, those binary values are the ASCII values of "column" and "value," respectively).

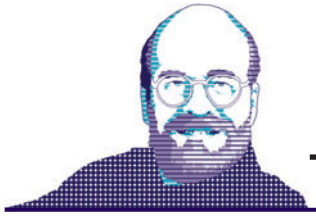
The Hardest Part Is Done

We're out of time, and Cassandra has only been installed. Specifically, a single-node Cassandra cluster is up and running, and nothing has been done to program against it yet. Fortunately, the hardest part of getting started with Cassandra has been completed. In the next installment, I'll start using .NET libraries to talk to Cassandra, get it to store some data from the .NET applications, pull it back, and then show how to set up a three-node cluster and get it up and running.

For now, though, 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 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 a Virtual World from Your Windows Phone

Until the time of Copernicus—and for many years after—people believed the universe was constructed of a series of concentric celestial spheres surrounding the Earth. Although that model of the universe has been abandoned, it's still convenient to employ the concept of a celestial sphere for identifying the location of objects in 3D space relative to ourselves as viewers.

A celestial sphere is particularly handy for programs that let you use a smartphone for viewing a world of virtual reality or augmented reality. With such programs, you hold the phone as if you're taking a photograph or video through the camera lens, but what you see on the screen might not have anything to do with the real world.

Such a program needs to determine its orientation in 3D space so that by sweeping the phone in arcs the user can pan through this virtual world. With the Motion sensor I described in the last installment of this column, Windows Phone is capable of providing the necessary orientation information.

How do we translate from the information provided by the Motion sensor to the celestial sphere? It's all about the coordinate system.

We're all familiar with *geographic coordinates* that allow us describe a location on the surface of our planet. Any point on the surface of the Earth can be denoted by two numbers: latitude and longitude, both of which are angles with a vertex in the Earth's center. Latitude is an angle relative to the equator: It's positive for locations north of the equator and negative for locations south. The latitude of the North Pole is 90° and the latitude of the South Pole is -90°. Longitude involves angles between great circles that pass through the two poles measured from the Prime Meridian, which is the line of longitude that passes through Greenwich, England.

We live not only on the surface of a sphere, but also at the center of a conceptual celestial sphere. Several coordinate systems can be used to denote locations on this celestial sphere, but the one I'll be focusing on is called the *horizontal coordinate system* because it's based on the horizon.

Horizontal Coordinates

Using your outstretched arm, point to any object you see around you. That object has a location on the celestial sphere. What is that location? Move your straight arm up or down so it becomes horizontal—that is, parallel to the surface of the Earth. The angle your arm swings through during this movement is called the *altitude*.

Code download available at archive.msdn.microsoft.com/mag201208TouchAndGo.

Positive values of altitude are above the horizon; negative values are below the horizon. An object located straight up from you has an altitude of 90°, also called the zenith, and an object straight down has an altitude of -90°, called the nadir.

Now swing your horizontal outstretched arm so it's pointing north. The angle your arm swings during this movement is called the azimuth. The altitude and azimuth together constitute a horizontal coordinate.

Notice that the horizontal coordinate gives you *no* information about how far away something is. During a solar eclipse, the sun and moon have the same horizontal coordinate. With any type of celestial coordinate system, everything is assumed to be on the interior surface of the celestial sphere.

The azimuth must be relative to a particular point on the compass. Most often, the azimuth is set at 0° for north, with increasing angles moving eastward. However, astronomers tend to set 0° at the south with increasing angles moving westward; at least that's how Jean Meeus sums it up in his classic book, "Astronomical Algorithms" (Willmann-Bell, 1998).

Horizontal coordinates are analogous to geographic coordinates, except the perspective is different. Instead of being on the surface of a sphere, you're at the center looking out. The azimuth is comparable to the longitude and the altitude is comparable to the latitude. Like circles of longitude, circles of azimuth are always great circles passing through the poles. Like circles of latitude, circles of altitude are always parallel to each other. The horizon plays the same role in horizontal coordinates as the equator in geographic coordinates.

Now pick up your Windows Phone and hold it so you're looking at the screen while the camera lens points away from you. The direction the camera lens is pointing has a particular altitude and azimuth. Although that horizontal coordinate is conceptually a location on the interior of the celestial sphere, it's also a direction from the viewpoint of the camera lens—mathematically, a three-dimensional vector.

As I've discussed in previous columns, the phone has an implicit coordinate system, where the positive Z axis extends out from the screen. That means the camera lens on the other side points in the direction of the 3D vector (0, 0, –1). As I demonstrated in the previous installment of this column (msdn.microsoft.com/magazine/jj190811), the Motion sensor in Windows Phone lets you obtain a 3D rotation matrix that describes how the Earth is rotated relative to the phone. To obtain a matrix that describes how the phone is rotated relative to the Earth, the matrix obtained from the Motion sensor must be inverted:

```
matrix = Matrix.Invert(matrix);
```

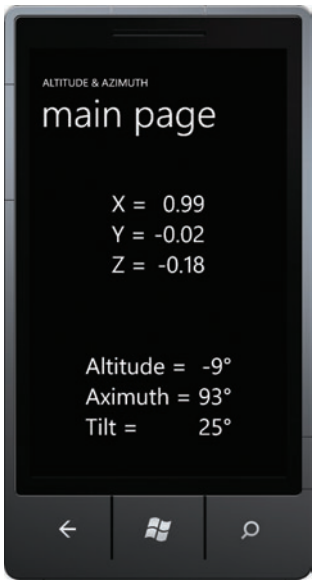


Figure 1 **The AltitudeAndAzimuth Display**

Figure 2 **The BigPicture Main Page**

Use this inverted matrix to rotate the (0, 0, –1) vector:

```
Vector3 vector = Vector3.Transform(new Vector3(0, 0, -1), matrix);
```

Now you have a 3D vector that describes the direction the camera lens is pointing. That vector needs to be converted to altitude and azimuth angles.

If the phone is held upright—that is, with the transformed vector horizontal to the surface of the Earth—the Z component is 0, and the problem reduces to the well-known conversion from two-dimensional Cartesian coordinates to polar coordinates. In C#, it's simply:

```
double azimuth = Math.Atan2(vector.X, vector.Y);
```

That's an angle in radians. Multiply by 180 and divide by π to convert to degrees.

This formula implies that north has an azimuth of zero, and values increase in an eastward direction.

If you prefer south to be zero with increasing values in a westward direction, shift the result by 180° by changing the sign of the X and Y components.

That formula for the azimuth is actually valid regardless of the Z component of the transform vector.

That Z component is the sine of the altitude. Because the altitude ranges only between negative and positive 90°, it can be calculated using the inverse sine function:

```
double altitude = Math.Asin(vector.Z);
```

Again, multiply by 180° and divide by π to convert radians to degrees.

However, we're still missing something, which you might recognize when you realize that we've translated a three-dimensional rotation matrix into a coordinate that has only two dimensions because it's confined to the interior surface of the celestial sphere.

What happens when you aim the phone in a particular direction described by a 3D vector, and then rotate the phone around the vector like an axis? The vector doesn't change, nor do the altitude and azimuth values, but the virtual reality scene on the phone's screen should rotate relative to the phone.

This extra motion is sometimes referred to as *tilt*. It's also an angle, but the calculation is a little more difficult than altitude and azimuth.

You can see that calculation in a HorizontalCoordinate structure I created that converts a Motion reading into altitude, azimuth and tilt, all in degrees. This structure is included in the AltitudeAndAzimuth project, which is among the downloadable code for this article. This program simply uses the Motion sensor to obtain the orientation of the phone, and then converts the information to horizontal coordinates. This project requires references to the Microsoft.Devices.Sensors assembly (for the Motion class) and the Microsoft.Xna.Framework assembly (for the 3D vector and matrix). The screen displays the transformed vector and the values from the HorizontalCoordinates structure. **Figure 1** shows the phone held approximately upright with the lens pointed approximately east, and tilted clockwise a bit.

Getting the Big Picture

Suppose you want to view an image that's much larger than the screen of your computer—or, in this case, your phone.

Traditionally, scrollbars have been involved. On a touchscreen, the scrollbars can be eliminated and the user can perform a similar scrolling operation using fingers.

But another approach is to conceptually wallpaper the interior of the celestial sphere with this large image and then view it by moving the phone itself. (Keep in mind that when you move the phone to view the image, you're not moving the phone left and right or up and down in a plane. The movement has to be along arcs so that the altitude and azimuth are changing.)

How large can such an image be so that it pans across the screen in a natural way as the phone moves?

An average Windows Phone screen is probably about 2 inches wide and 3.33 inches tall. If you hold the phone 6 inches from your face, some simple trigonometry reveals that the phone occupies a field of view about 19° degrees wide and 31° tall. Holding the phone in landscape mode, these two fields of view are slices from the total azimuth of 360° degrees and altitude of 180°. Very roughly, then, the phone's screen held 6 inches from your face in landscape mode occupies about 10 percent of the total field of view horizontally and vertically.

Or think of it this way: If you want to use your phone in portrait mode to pan over the surface of a bitmap, that bitmap can be somewhere in the region of 8000 pixels wide and 4800 pixels high.

That's the idea of the BigPicture project, which contains links for downloading eight images (a mix of paintings, photographs, documents and drawings, mostly from Wikipedia), the largest of which is 5649 pixels wide and 4000 pixels high. You can easily add other images by editing an XML file, but based on my experience using the PictureDecoder.DecodeJpeg method, you're likely to encounter out-of-memory exceptions if you go much larger.

Background File Transfers

Considering that most of the image files referenced by BigPicture are more than 2MB in size and one of them is 19MB, this seemed an ideal opportunity to make use of the facility added to Windows Phone to download files in the background.

Figure 3 **BigPicture Showing One Small Part of a Large Painting**

Figure 4 **BigPicture Showing an Entire Large Painting**

In the BigPicture program, most of MainPage is devoted to maintaining a ListBox that lists the available files, and downloading them to isolated storage. **Figure 2** shows the program with some images already downloaded (which are shown as thumbnails), one download in progress and others not yet downloaded.

To use the background file transfer, you create an object of type BackgroundTransferRequest, passing to it the URL of the external file and the URL of a location in isolated storage within the /shared/transfers directory. You can then obtain changes in status and progress via events while the program is running, and you can enumerate the active requests when your program starts up again.

When the BigPicture program starts up, MainPage searches isolated storage for any images that might have been previously downloaded. I discovered that files are downloaded directly to the filename you specify, and not to a temporary file with some other filename. This means that my program was encountering files that had not yet been fully downloaded, or whose downloads might have been canceled. I fixed several bugs in my program by using the /shared/transfers directory only for downloading files and not for permanent storage. When a download completes, the program moves that file to another directory and creates a thumbnail in yet another directory. For convenience, all three files have the same name but are distinguished by the directory in which they're found.

When a file has been downloaded by BigPicture, you can tap the item in the ListBox and the program navigates to ViewPage, which is the real heart of the program.

Viewing the Big Image

ViewPage has two viewing modes, which you can alternate between by tapping on the screen. An animation takes you from one mode to the other.

In the normal mode, shown in **Figure 3**, the image is displayed in its pixel size, conceptually stretched to the interior of a celestial sphere. You navigate around the image by changing the orientation of the phone, conceptually pointing the phone toward the area of the celestial sphere you want to view. (It might help if you stand up and turn your whole body in different directions, and point the phone up and down as well.)

When you tap the phone, you shift to the zoom-out mode. The entire image is displayed unrotated in portrait mode, as shown in **Figure 4**. A rectangle displays the portion of the image that's viewable in the normal mode. In this example, that rectangle is near the lower-right corner.

What happens at the edges? Because the bitmap is conceptually stretched to the interior of a celestial sphere, when you move the phone to the right beyond the right edge of the bitmap, you should then encounter the left edge. However, the layout system in Silverlight doesn't wrap around in this way. If the program allowed opposite edges of a large bitmap to be visible, then two Image elements would be required. At the point where all four corners meet, four Image elements would be required.

I nixed that concept. Beyond the right edge of the bitmap is a gap equal to the maximum dimension of the phone's display, and then the left edge appears. You'll never see both edges in the display. This also solves the problem of what to do at the poles, where theoretically the top and bottom of the painting should be compressed to a point.

Figure 5 shows most of the XAML file for ViewPage. The Image element displays the bitmap itself, of course, and the None setting for the Stretch property indicates that it's to be displayed in its pixel size. Normally a large image would be cropped by the layout system

Figure 5 **The XAML File for the BigPicture Image Viewing Page**

```
<phone:PhoneApplicationPage ... >

  <Grid x:Name="LayoutRoot" Background="Transparent">
    <Canvas>
      <Grid>
        <Image Name="image" Stretch="None" />

        <Border Name="outlineBorder"
                BorderBrush="White"
                HorizontalAlignment="Left"
                VerticalAlignment="Top">

            <Rectangle Name="outlineRectangle"
                       Stroke="Black" />

            <Border.RenderTransform>
              <CompositeTransform x:Name="borderTransform" />
            </Border.RenderTransform>
        </Border>

        <Grid.RenderTransform>
          <CompositeTransform x:Name="imageTransform" />
        </Grid.RenderTransform>
      </Grid>
    </Canvas>

    <TextBlock Name="titleText"
               Style="{StaticResource PhoneTextNormalStyle}"
               Margin="12,17,0,28" />

    <TextBlock Name="statusText"
               Text="creating image..."
               HorizontalAlignment="Center"
               VerticalAlignment="Center" />
  </Grid>
</phone:PhoneApplicationPage>
```

Figure 6 **Much of the Transform Math for BigPicture**

```
public partial class ViewPage : PhoneApplicationPage
{
  ...
  void OnLoaded(object sender, RoutedEventArgs args)
  {
    // Save the screen dimensions
    screenWidth = this.ActualWidth;
    screenHeight = this.ActualHeight;
    maxDimension = Math.Max(screenWidth, screenHeight);

    // Initialize some values
    outlineBorder.Width = screenWidth;
    outlineBorder.Height = screenHeight;
    borderTransform.CenterX = screenWidth / 2;
    borderTransform.CenterY = screenHeight / 2;

    // Load the image from isolated storage
    ...
    // Save image dimensions
    imageWidth = bitmap.PixelWidth;
    imageHeight = bitmap.PixelHeight;
    ...
    zoomInScale = Math.Min(screenWidth / imageWidth, screenHeight / imageHeight);
    UpdateImageTransforms();
    ...
  }
  ...
  void OnMotionCurrentValueChanged(object sender,
                                   SensorReadingEventArgs<MotionReading> args)
  {
    ...
    // Get the rotation matrix & convert to horizontal coordinates
    Matrix matrix = args.SensorReading.Attitude.RotationMatrix;
    HorizontalCoordinate horzCoord = HorizontalCoordinate.FromMotionMatrix(matrix);

    // Set the transform center on the Image element
    imageTransform.CenterX = (imageWidth + maxDimension) *
      (180 + horzCoord.Azimuth) / 360 - maxDimension / 2;
    imageTransform.CenterY = (imageHeight + maxDimension) *
      (90 - horzCoord.Altitude) / 180 - maxDimension / 2;

    // Set the translation on the Border element
    borderTransform.TranslateX = imageTransform.CenterX - screenWidth / 2;
    borderTransform.TranslateY = imageTransform.CenterY - screenHeight / 2;

    // Get rotation from Tilt
    rotation = -horzCoord.Tilt;
    UpdateImageTransforms();
  }

  static void OnInterpolationFactorChanged(DependencyObject obj,

DependencyPropertyChangedEventArgs args)
  {
    (obj as ViewPage).UpdateImageTransforms();
  }

  void UpdateImageTransforms()
  {
    // If being zoomed out, set scaling
    double interpolatedScale = 1 + InterpolationFactor * (zoomInScale - 1);
    imageTransform.ScaleX =
    imageTransform.ScaleY = interpolatedScale;

    // Move transform center to screen center
    imageTransform.TranslateX = screenWidth / 2 - imageTransform.CenterX;
    imageTransform.TranslateY = screenHeight / 2 - imageTransform.CenterY;

    // If being zoomed out, adjust for scaling
    imageTransform.TranslateX -= InterpolationFactor *
      (screenWidth / 2 - zoomInScale * imageTransform.CenterX);
    imageTransform.TranslateY -= InterpolationFactor *
      (screenHeight / 2 - zoomInScale * imageTransform.CenterY);

    // If being zoomed out, center image in screen
    imageTransform.TranslateX += InterpolationFactor *
      (screenWidth - zoomInScale * imageWidth) / 2;
    imageTransform.TranslateY += InterpolationFactor *
      (screenHeight - zoomInScale * imageHeight) / 2;

    // Set border thickness
    outlineBorder.BorderThickness = new Thickness(2 / interpolatedScale);
    outlineRectangle.StrokeThickness = 2 / interpolatedScale;

    // Set rotation on image and border
    imageTransform.Rotation = (1 - InterpolationFactor) * rotation;
    borderTransform.Rotation = -rotation;
  }
}
```

to the size of the display, and you wouldn't be able to pan around the rest of the image. But putting everything inside a Canvas tricks the layout system into rendering the whole object. The Border with the embedded Rectangle is the rectangle visible in the zoomed-out mode, but it's also visible hugging the inside of the screen in the normal mode. The CompositeTransform named imageTransform applies to both the Image and the Border. The other CompositeTransform named borderTransform applies only to the Border.

The codebehind file starts a Motion sensor going and then applies the rotation matrix to create a HorizontalCoordinate object that it uses to set the properties of these two transforms. The ViewPage class also defines an InterpolationFactor dependency property that's the target of an animation to transition between the two viewing modes. As InterpolationFactor is animated from 0 to 1, the view transitions between the normal and the zoom-out.

Figure 6 shows most of the math involved. One of the most important calculations occurs when the Motion sensor is updated. This is the calculation of the CenterX and CenterY properties of the CompositeTransform for the Image, and it's where the altitude and azimuth come into play. Although this transform center is the point around which scaling and rotation occurs, further calculations put this point in the center of the display in the normal viewing mode. The rectangular border is also aligned with this point.

When the Azimuth is 0 (phone facing north) and the Altitude is 0 (upright), the CenterX and CenterY properties are set to the center of the bitmap. Notice the inclusion of the maxDimension value so that these CenterX and CenterY properties can be set to values outside the bitmap. This allows for the padding when you sweep past the edges.

Most of the remainder of the calculations occur during the UpdateImageTransforms method, which is called when the Motion sensor reports a new value, or when the InterpolationFactor property changes during transitions. Here's where the scaling and translation of the Image transform occurs, as well as rotation.

If you're interested in understanding the interaction of these transforms, you might want to clean them up by eliminating all the interpolation code. Examine the simplified formulas when InterpolationFactor is 0 and when it's 1, and you'll see that they're actually quite straightforward. ■

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*

DAVID S. PLATT

Whither Windows 8 Hardware

The Microsoft phone team understands that they need good hardware in order for their software to succeed. The Microsoft tablet team is starting to understand this, too.

You don't need me to tell you about the success of the Apple iPad, expected to sell its 100-millionth unit this year (source: bit.ly/yyJrIo). I'll bet more than half of my readership owns one, as do I.

Most Windows hardware vendors are trying to challenge the iPad with a hybrid PC, such as the Lenovo Yoga or the Samsung prototype Microsoft gave out at the BUILD conference last year. But compared to the iPad, both those machines are heavy, hot and expensive, and have poor battery life. Compared to a notebook PC, they're underpowered and hard to type on. They have the drawbacks of both form factors and the advantages of neither. As a Texan student of mine once observed, "There ain't nothing in the middle of the road 'cept yellow lines and squashed armadillos."

As a Texan student of mine once observed, "There ain't nothing in the middle of the road 'cept yellow lines and squashed armadillos."

I'd like to take a moment to formally coin the term "armadillo" to denote a technology product that fails because its functionality falls between two successful niches, offering the drawbacks of both but the advantages of neither. As in, "The Samsung Series 7 Slate is too hot, heavy and expensive for a tablet, and too weak and hard to type on for a notebook. What an armadillo!" Oxford English Dictionary, please take note.

Not content with these offerings, Microsoft is now attacking Apple directly with the just-announced Surface tablet PC. The size, weight and display are roughly equivalent to the iPad, and the Surface has a foldable keyboard, USB slot and SD card slot, which the iPad does not.

Assuming that Microsoft can manufacture this device with reasonable quality, the key to its success will be price. Lament it or envy it, but you can't deny it: Customers are willing to pay more for Apple products than for those of other vendors. Microsoft acknowledges, nay, trumpets this fact in its advertising, as you can see here: bit.ly/MW140X.

The Apple business model resembles that of Whole Foods Market, an upscale grocery store chain featuring natural and organic foods. The shopping experience is part of the enjoyment, and you pay for it in the higher prices of items.

The Microsoft business model resembles that of Safeway, a not-particularly elegant supermarket chain common in Seattle. It's nowhere near as nice, but it's a whole lot cheaper. That's what most customers choose, and that's why Safeway has much higher gross sales than Whole Foods—lower margins, but the volume more than compensates for it. To keep the Safeway model going, Microsoft has to sell the Surface for at least 25 percent less than the iPad.

If Microsoft really wants to light a fire in the industry, it should get a Windows 8 tablet onto store shelves this Christmas for the $199 price of the Amazon Kindle Fire (as Google is now trying with its new Nexus 7). Having used Android, I can tell you that Windows 8 and the Metro-style UX is better. Hitting that price point, even with lower functionality (no keyboard, a stamped-metal case instead of vapor-deposited magnesium and so on), would capture Microsoft major market share.

That's what I think Microsoft has in mind for its alliance with Barnes & Noble (source: bit.ly/L3tM1I). I'm guessing that Microsoft wants the Nook tablet to run Windows 8. The increased sales of Barnes & Noble media could then allow the company to subsidize the consumer price somewhat, as phone providers do—and as Amazon does for the Kindle.

Can Microsoft do it? I don't know. The market will give the answer, as it should, and as it always does.

So here's the brawl I promised you in my November 2011 column (msdn.microsoft.com/magazine/hh547110). Apple versus Google versus Microsoft, with retail vendors Amazon and Barnes & Noble jumping in, and Facebook trying to figure out where it fits. I love my ringside seat, and I'm ever ready to grease the stairs or pour oil on troubled fires for my own amusement and yours. Damn, it's a great time to be alive and working in this industry. ■

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.*